MICRO-ORDINATEURS

LANGAGE MACHINE, TRUCS ET ASTUCES SUR ZX 81

Gabriel NOLLET

EYROLLES

LANGAGE MACHINE TRUCS ET ASTUCES SUR ZX 81

DANS LA MÊME COLLECTION

SCHOMBERG - Le Basic Universel.
SCHOMBERG - Micro-ordinateurs : Comment ça marche ?
HERNANDEZ - Pascal par l'exemple.
NOLLET - La conduite du ZX 81.
PELLIER - La conduite du TRS 80.
LADEVIE - Votre gestion avec BASIC sur micro-ordinateur.
QUEINNEC - Langage d'un autre type : LISP.
PELLIER - Programmez vos jeux d'action rapide sur TRS 80.
ASTIER - La conduite du l'APPLE II
Tome 1 : le Basic de l'APPLE II
Tome 2 : le système graphique et l'assembleur de l'APPLE II.
MONTEIL - L'assembleur facile du 6502.
LEPAPE - L'assembleur facile du Z 80.
OROS et - ZX 81 à la conquête des jeux.
PERBOST - CASSETTE - ZX 81 à la conquête des jeux.
DAX - CP/M et sa famille.
NOLLET - Langage machine, trucs et astuces sur ZX 81
BICKING - La conduite du PC 1211 (ou TRS 80 pocket).
TEJA - Apprenez à parler à votre ordinateur.
MONTEIL - La conduite du VIC 20.
SAGUEZ - Contrôle. Commande par micro-ordinateur.
PLOUIN - La conduite de l'IBM-PC.
PELLIER - Tout sur les disques du TRS 80 modèles I et II.
GROS - La conduite du PC 1500

« La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservés à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'article 40) ».

« Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal ».

© Éditions EYROLLES, 1983

LANGAGE MACHINE TRUCS ET ASTUCES SUR ZX 81

par

Gabriel NOLLET

Collection animée
par Richard SCHOMBERG

DEUXIÈME ÉDITION
Nouveau tirage

EYROLLES

61, boulevard Saint-Germain — 75005 Paris
1983

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service «Presse», Éditions EYROLLES
61, Boulevard Saint-Germain,
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

Avant-Propos

Vous pratiquez depuis quelques temps le langage BASIC et vous le maitrisez maintenant parfaitement. Les nombreux jeux que vous aviez réalisés avec vos amis pour le ZX 81 et qui à l'origine avaient suscité tant d'enthousiasme de votre part, vous laissent maintenant complètement indifférents !... Devant ce mal insidieux, il ne vous reste alors plus qu'une seule solution: Faire du langage machine !

"Langage machine, trucs et astuces sur ZX1 " s'adresse à tous ceux qui possèdent déjà une connaissance du langage machine du microprocesseur Z 80 ou d'un autre microprocesseur.

Ceux qui n'ont jamais abordé le langage machine liront avec profit le livre "l'assembleur facile du Z 80" paru dans la même collection. La lecture de cet ouvrage de base leur donnera une parfaite compréhension de tous les principes de la programmation en langage machine ceux-ci étant illustrés par de nombreux exemples s'appliquant plus particulièrement au microprocesseur Z 80 (qui est celui utilisé dans votre ZX 81 !)

Nous vous conseillons aussi, pour bien comprendre le chapitre sur la scrutation du clavier, de vous reporter au livre "Micro-ordinateur: comment ça marche ?" de la même collection.

Après avoir lu "Langage machine, trucs et astuces sur ZX 81 ", vous saurez comment faire pour générer une instruction REM de 1,

2, 3..., 10 K octets (et même plus si vous le désirez !), comment scruter le clavier, obtenir des graphiques animés extrêmement rapides ! Vous y apprendez aussi pourquoi il faut faire attention lorsque l'on utilise le code héxadécimal 7E dans une instruction REM !

"Langage machine, trucs et astuces sur ZX 81" vous donnera aussi le listage d'un programme d'aide à la mise au point des programmes écrits en langage machine. Ce programme, appelé moniteur, vous permettra de rentrer directement le code hexadécimal de votre programme écrit en langage machine, de lister des zones de la mémoire, de lire les contenus des registres du microprocesseur Z 80 et d'autres choses encore faisant de ce moniteur un outil indispensable à ceux qui désirent pratiquer le langage machine sur le ZX 81.

Dans tout ce qui suit la lettre H, placée immédiatement après un nombre, indiquera que ce nombre est exprimé en hexadécimal, ainsi 16 H correspond à la valeur décimale 22.

Table des matières

AVANT-PROPOS VII

INTRODUCTION XI

1. Rappel des instructions 1
1.1. Les registres 1
1.2. Les instructions du Z 80 2

2. Où et comment stocker un programme écrit en langage machine 37
2.1. Méthodes pour stocker un programme écrit en langage machine 37
2.2. Génération d'une instruction REM de plusieurs centaines d'octets 48
2.3. Comment sauvegarder un programme en langage machine situé dans le haut de la mémoire 54

3. Programme d'aide à la mise au point de programmes écrits en langage machine 56
3.1. Programme d'aide à la mise au point 57
3.2. Utilisation du programme 61

4. Comment scruter le clavier 72
4.1. Fonctionnement du sous-programme de la ROM 72
4.2. Comment retrouver le code du caractère correspondant à la touche appuyée 79
4.3. Autre méthode de scrutation du clavier 82

5. Maîtriser le buffer d'affichage 87
5.1. Comment afficher des caractères 87

5.2. Travailler directement dans le buffer d'affichage 93

6. Faites vos jeux !... 103
6.1. Quelques méthodes graphiques 103

7. Utiliser un assembleur 133

ANNEXES 137
1. **Fonctionnement du programme d'aide à la mise au point et améliorations possibles** 137
2. **Fonctionnement du clavier du ZX 81** 152
3. **Instructions du Z 80 et leurs codes** 154
4. **Code des instructions du Z 80** 159
5. **Code des caractères du SINCLAIR ZX 81** 165
6. **Adresses utiles** 169

Introduction

Le circuit principal de votre SINCLAIR ZX 81 est le microprocesseur Z 80. C'est lui qui réalise et exécute toutes les instructions demandées. Toutefois celui-ci ne peut pas être programmé directement *par une instruction BASIC ; en effet, le Z 80 ne reconnaît que les codes hexadécimaux des instructions du langage machine. Comment une instruction BASIC est-elle alors exécutée ? A chaque instruction ou commande BASIC est associé un ou plusieurs programmes écrit en langage machine. Dès qu'une instruction est reconnue, le programme en langage machine qui lui correspond est exécuté. Cet ensemble des programmes écrits en langage machine constitue ce qu'on appelle la ROM BASIC. Pour le ZX 81 cette ROM BASIC occupe 8 K octets.*

Ces différents programmes s'assurent entre autre de la validité des paramètres de l'instruction BASIC et génèrent un code d'erreur si besoin est. Pour un programme écrit en langage machine, aucune vérification de ce genre n'est effectuée et les instructions sont exécutées en séquence. Ainsi un débranchement à une mauvaise instruction dans un programme écrit en BASIC ne sera pas trop grave et sera vite détecté alors que la même erreur dans un programme écrit en langage machine pourra avoir (et aura sûrement !...) des répercussions catastrophiques. Il apparaît donc qu'un programme écrit en langage machine s'exécutera beaucoup plus rapidement qu'un pro-

gramme BASIC mais il sera, par contre, beaucoup plus difficile à mettre au point.

"Langage machine, trucs et astuces sur ZX 81" vous apportera une aide efficace pour réaliser et mettre au point vos programmes en langage machine.

- *Dans le premier chapitre nous rappelons les instructions du microprocesseur Z 80 en les accompagnant d'un exemple.*
- *Au chapitre 2 vous trouverez différentes méthodes pour stocker un programme écrit en langage machine.*
- *Le chapitre 3 vous donne le listage d'un programme d'aide à la mise au point et vous indique comment utiliser cet outil indispensable !*
- *Les chapitres 4 et 5 vous apprendront respectivement à scruter le clavier et à maitriser le buffer d'affichage.*
- *Le chapitre 6 est entièrement consacré au graphique. Un jeu de "mur de briques" y est détaillé complètement.*
- *Au chapitre 7 nous aborderons l'utilisation d'un assembleur.*

1

Rappel des instructions du Z 80

Dans ce chapitre nous reprenons en les accompagnant d'un exemple les instructions du Z 80. Pour chacun des exemples étudiés vous trouverez les contenus des registres après l'exécution de l'instruction considérée. (Les contenus des registres ont été obtenus avec le moniteur détaillé au chapitre 3).

Pour chacun des exemples, nous avons souligné les registres concernés par l'instruction présentée.

Chaque exemple est un petit sous-programme écrit en langage machine et se termine donc par une instruction RET.

1.1. LES REGISTRES

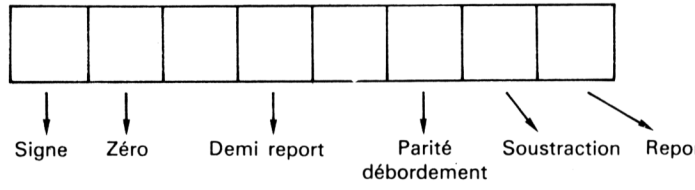
Le microprocesseur Z 80 possède deux séries de registres. Les registres primaires et les registres secondaires.

Il existe 14 registres primaires. Le registre A ou accumulateur, le registre indicateur ou F, les registres B, C, D, E, H et L, les registres d'index IX et IY, le pointeur de la pile ou registre SP, le compteur ordinal ou registre PC, le registre d'interruption I et le registre de rafraîchissement de mémoire R. Certains de ces registres sont utilisés pour le fonctionnement du ZX 81 notamment les registres I, R, IX et IY.

Remarque: Si vous travaillez en **mode SLOW n'utilisez pas le registre**

d'index IX. En effet celui-ci est utilisé pour gérer l'affichage. Si vous modifiez le contenu de ce registre votre ZX 81 se "plantera" et vous risquez fort de devoir débrancher et rebrancher l'alimentation de votre ZX 81 pour réobtenir un fonctionnement normal.

Il existe huit registres secondaires. Ces registres appelés A', F', B', C', D', E', H' et L' peuvent être utilisés pour sauvegarder le contenu des registres primaires. Toutefois, comme pour le registre IX, les registres secondaires A' et F' sont utilisés pour gérer l'affichage. Ne les utilisez donc pas. La structure du registre indicateur est la suivante.



1.2. LES INSTRUCTIONS DU Z 80

ADC

ADC A, r

r est un registre, une constante, ou une position mémoire adressée par les registres HL, IX ou IY.

Addition de l'opérande r avec l'accumulateur et le report. Le résultat est stocké dans l'accumulateur.

Exemple: (avec le report à 1). Addition des registres A et B.

LD	A, 03 H	3E	03	
LD	B, 0A H	06	0A	
SCF		37		
ADC	A, B	88		
RET		C9		

Contenus des registres après exécution de l'instruction ADC.

```
?R
 F=08  A=0E  C=00  B=0A  E=45  D
=40 HL=411A IX=028F IY=4000
```

ADC HL, rp

rp est une paire de registres (BC, DE, HL ou SP).

Addition de la paire de registres HL et de la paire de registres rp avec le report. Le résultat est rangé dans la paire de registres HL.

Exemple : (avec le report à 1) addition des paires de registres HL et BC.

LD	HL, 1234 H	21	34	12
LD	BC, 0005 H	01	05	00
SCF		37		
ADC	HL, BC	ED	4A	
RET		C9		

```
?R
 F=00  A=63  C=05  B=00  E=45  D
=40 HL=123A IX=028F IY=4000
```

ADD

ADD r1, r2

r1 : est l'accumulateur, la paire de registres HL, le registre IX ou le registre IY.

r2 : est un registre, un emplacement mémoire adressé par la paire de registres HL ou les registres d'index IX et IY, une constante ou une paire de registres.

Addition des contenus des opérandes r1 et r2. Le résultat est rangé dans l'opérande r1.

Exemple : ajouter 3 au contenu de l'accumulateur et multiplier par 2 le contenu de la paire de registres HL.

LD	A, 45 H	3E	45	
ADD	03	C6	03	
LD	HL, 1234 H	21	34	12
ADD	HL, HL	29		
RET		C9		

```
?R
 F=20  A=48  C=00  B=00  E=45  D
=40 HL=2468 IX=028F IY=4000

?
```

AND

AND r

r est un registre, une constante ou une position mémoire adressée par les registres HL, IX ou IY.

La fonction AND réalise l'opération logique ET entre l'accumulateur et l'opérande r. Le résultat est rangé dans l'accumulateur.

Exemple: La séquence d'instructions suivante masque les quatre bits de poids forts de l'accumulateur.

LD	A, 75 H	3E	75
AND	0FH	E6	0F
RET		C9	

```
?R
 F=14  A=05  C=00  B=00  E=45  D
=40 HL=411A IX=028F IY=4000

?
```

Remarque: Cette instruction est souvent utilisée pour positionner les indicateurs notamment pour remettre l'indicateur Report (C) à zéro.

Exemple: La séquence d'instructions suivante mettra l'indicateur zéro à 1 si le contenu de l'accumulateur est nul.

LD	A, 1	3E	01
AND	A	A7	
RET		C9	

```
?R
 F=10  A=01  C=00  B=00  E=45  D
=40 HL=411A IX=028F IY=4000

?
```

Le contenu hexadécimal du registre Flag est 10. L'indicateur Z (zéro) n'est donc pas positionné. Reportez-vous à la structure du registre indicateur au début de ce chapitre.

Reprenons cet exemple en forçant l'accumulateur à zéro. Le contenu hexadécimal du registre Flag devient 54. L'indicateur Z est bien positionné à 1.

```
?R
 F=54   A=00   C=00   B=00   E=45   D
=40  HL=411A  IX=028F  IY=4000

?
```

Remarque: Avec l'instruction AND l'indicateur C (report) est toujours forcé à zéro.

BIT

BIT b, r

r est un registre ou une position mémoire adressée par les registres HL, IX ou IY.

b est un nombre compris entre 0 et 7.

L'instruction BIT b, r teste le bit b de l'opérande r. L'indicateur Z sera forcé:

— à zéro si le bit testé est à un,
— à un si le bit testé est à zéro.

Exemple: Dans cette exemple nous chargerons l'octet situé à l'adresse décimale 16676 avec la valeur hexadécimale 55

(55 H = 01010101)

Après avoir testé le bit 7 nous vérifierons en regardant le contenu du registre F que l'indicateur Z a bien été mis à 1.

LD	HL, 16676	21	24	41
LD	(HL), 55 H	36	55	
BIT	7, (HL)	CB	7E	
RET		C9		

```
?R
 F=54   A=61   C=00   B=00   E=45   D
=40 HL=4124  IX=028F  IY=4000
```

CALL

CALL adr

adr est une adresse.

L'instruction CALL permet de se brancher à un sous programme dont la première instruction est située à l'adresse adr. L'adresse de retour est sauvegardée dans la pile.

CALL cc, adr

cc est une condition.

Cette instruction est un appel conditionnel de sous-programme. Si la condition cc est réalisée alors le sous-programme sera appelé sinon l'appel est ignoré et l'instruction suivante est exécutée.

Exemple: Dans l'exemple suivant le sous-programme appelé force le contenu de l'accumulateur avec la valeur – 1 (FF en hexadécimal). Ce sous-programme, situé à l'adresse décimale 16675, ne sera appelé que si le flag Z est à 1.

Sous-programme.

LD	A, – 1	3E	FF	
RET		C9		
(Programme principal .)				
LD	HL, 16700	21	3C	41
LD	(HL), 55 H	36	55	
LD	A, 32 H	3E	32	
BIT	7, (HL)	CB	7E	
CALL	Z, 16675	CC	23	41
RET		C9		

Dans ce cas l'appel du sous-programme est effectué.

```
?R
 F=54   A=FF   C=00   B=00   E=45   D
=40 HL=413C  IX=028F  IY=4000

?
```

Remplacez l'instruction BIT 7, (HL) par BIT 6, (HL) — code hexadécimal CB 76 — l'appel du sous-programme n'est plus effectué.

```
?R
 F=10  A=32  C=00  B=00  E=45  D
=40 HL=413C IX=028F IY=4000
```

CCF

L'instruction CCF complémente l'indicateur de report. (*ie* celui-ci est mis à zéro s'il était à un et inversement).

Exemple: Dans cet exemple nous mettons l'indicateur de report à zéro. Pour cela nous le forcerons d'abord à 1. (instruction SCF).

SCF		37			
CCF		3F			
RET		C9			

État des registres avant l'exécution de l'instruction CCF.

```
?R
 F=09  A=5B  C=00  B=00  E=45  D
=40 HL=411A IX=028F IY=4000

?
```

État des registres après exécution de l'instruction CCF. Vous pouvez constater que le contenu du registre Flag est devenu 18

```
?R
 F=18  A=5D  C=00  B=00  E=45  D
=40 HL=411A IX=028F IY=4000
```

CP

CP r

r est un registre, une constante ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction CP r compare le contenu de l'accumulateur au contenu de l'opérande r. La comparaison est effectuée en soustrayant le

contenu de l'opérande r au contenu de l'accumulateur. Toutefois le résultat de la soustraction n'est pas conservé ; le contenu de l'accumulateur sera donc inchangé.

Remarque : L'indicateur report (C) sera mis à 1 si le contenu de l'accumulateur est inférieur au contenu de l'opérande r.

L'indicateur zéro (Z) sera mis à 1 si le contenu de l'accumulateur est égal au contenu de l'opérande r.

Exemple : Comparaison de l'accumulateur à la valeur hexadécimale 45. Vous remarquerez que l'indicateur de report (C) a été mis à 1 après la comparaison.

AND	A	A7		
LD	A, 10 H	3E	10	
CP	45 H	FE	45	
RET		C9		

```
?R
 F=93  A=10  C=00  B=00  E=45  D
=40 HL=411A IX=028F IY=4000
```

CPD

L'instruction CPD compare le contenu de l'accumulateur avec le contenu de la position mémoire adressée par la paire de registres HL. Les paires de registres HL et BC sont ensuite décrémentées de 1.

Exemple : Comparaison de l'accumulateur avec le contenu de la position mémoire située à l'adresse décimale 16700.

LD	HL, 16700	21	3C	41
LD	(HL), 15 H	36	15	
LD	A, 15 H	3E	15	
LD	BC, 0004H	01	04	00
CPD		ED	A9	
RET		C9		

Contenu des registres avant exécution de l'instruction CPD.

```
?R
 F=28  A=15  C=04  B=00  E=45  D
=40 HL=413C IX=028F IY=4000
```

Contenu des registres après exécution de l'instruction CPD. Les paires de registres HL et BC ont été décrémentées de 1. Remarquez que l'indicateur zéro (Z) a été mis à un, les contenus de l'accumulateur et de la position mémoire 16700 étant égaux.

```
?R
 F=46   A=15   C=03   B=00   E=45   D
=40  HL=413B  IX=028F  IY=4000

?
```

CPDR

L'instruction CPDR est analogue à l'instruction CPD. Toutefois l'instruction sera réexécutée jusqu'à ce que:

— Le contenu de la paire de registres BC soit nul,

ou

— le contenu de l'accumulateur soit égal au contenu de la position mémoire adressée par la paire de registre HL.

Exemple: Reprenons l'exemple précédent:

LD	HL, 16700	21	3C	41
LD	(HL), 15 H	36	15	
LD	A, 15 H	3E	15	
LD	BC, 0004 H	01	04	00
CPDR		ED	B9	
RET		C9		

Les contenus de l'accumulateur et de la position mémoire 16700 sont égaux. L'instruction CPDR n'est exécutée qu'une seule fois. A la fin de l'exécution du programme les contenus hexadécimaux des paires de registres HL et BC sont respectivement 413B et 0003.

```
?R
 F=46   A=15   C=03   B=00   E=45   D
=40  HL=413B  IX=028F  IY=4000
```

Relançons ce programme en forçant le contenu de l'accumulateur à zéro. (On s'assurera auparavant que la zone mémoire 16697 à 16700 ne

contient pas le code 00). L'instruction CPDR est exécutée 4 fois. A la fin de l'exécution du programme, les contenus hexadécimaux des paires de registres HL et BC seront respectivement 4138 et 0000.

```
?R
 F=B2   A=00   C=00   B=00   E=45   D
=40  HL=4138  IX=028F  IY=4000

?
```

CPI

L'instruction CPI est analogue à l'instruction CPD. Toutefois la paire de registres HL sera incrémentée au lieu d'être décrémentée. La paire de registres BC est toujours décrémentée.

Exemple :

LD	HL, 16700	21	3C	41
LD	(HL), 15 H	36	15	
LD	A, 15 H	3E	15	
LD	BC, 0004 H	01	04	00
CPI		ED	A1	
RET		C9		

Contenu des registres après exécution de l'instruction CPI.

```
?R
 F=45   A=15   C=03   B=00   E=45   D
=40  HL=413D  IX=028F  IY=4000

?
```

CPIR

L'instruction CPIR est analogue à l'instruction CPDR. Toutefois la paire de registres HL sera incrémentée au lieu d'être décrémentée. La paire de registres BC est toujours décrémentée.

Exemple:

LD	HL, 16700	21	3C	41
LD	(HL), 15 H	36	15	
LD	A, 00 H	3E	00	
LD	BC, 0004 H	01	04	00
CPIR		ED	B1	
RET		C9		

```
?R
 F=B2   A=00   C=00   B=00   E=45   D
=40  HL=4140  IX=028F  IY=4000

?
```

CPL

L'instruction CPL complémente le contenu de l'accumulateur

Exemple:

LD	A, 55 H	3E	55
CPL		2F	
RET		C9	

Contenu des registres après exécution de l'instruction CPL.

```
?R
 F=3A   A=AA   C=00   B=00   E=45   D
=40  HL=411A  IX=028F  IY=4000  SP=7
FEA

?
```

DAA

Ajustement décimal de l'accumulateur.

L'instruction DAA est utilisée lorsque l'on travaille en "décimal codé binaire". (Rappel: en décimal codé binaire le nombre décimal 15, par exemple, sera représenté sur un octet de la manière suivante:

0 001	0 1 0 1
1	5

c'est-à-dire 15 en hexadécimal et non 0F comme on aurait pu s'y attendre).

Supposons que l'on désire additionner deux nombres mémorisés en décimal codé binaire, 15 et 8 par exemple. Le résultat donné par le Z 80 sera 1D en hexadécimal alors que nous voulions, nous,obtenir 23 comme résultat. L'instruction DAA, utilisée après l'addition, "transformera" la valeur 1D en 23. Pour cela l'instruction DAA opère de la manière suivante :

— Si les quatre bits de poids faibles de l'accumulateur donnent une valeur strictement supérieur à 9 ou si l'indicateur de demi report est à 1 alors, ajouter six au contenu de l'accumulateur.

— Si les quatre bits de poids forts de l'accumulateur donnent une valeur strictement supérieur à 9 ou si l'indicateur de report est à 1 alors, ajouter six au contenu des quatres bits de poids forts de l'accumulateur.

Exemple : Additionnons les deux nombres 15 et 8 mémorisés en binaire codé décimal ; nous aurons :

```
      00010101 :     15 H
    + 00001000 :     08 H
  ------------     ------
(A) = 00011101 :     1D H
```

avec les indicateurs (C) Report = 0 et (H) demi report = 0

0D H est strictement supérieur à 9 : ajoutons donc 6 au contenu de l'accumulateur : celui-ci devient donc :

```
    00011101 :     1D H
  + 00000110 :     06 H
  ----------     ------
  = 00100011 :     23 H
```

avec les indicateur C = 0, H = 1

L'indicateur Report (C) est à zéro et le contenu des quatre bits de poids forts (2) est inférieur à 9. Le résultat est donc 23. Nous pouvons vérifier ceci avec le petit programme suivant.

LD	A, 15 H	3E	15		
ADD	A, 08 H	C6	08		
DAA		27			
RET		C9			

Contenu des registres avant exécution de l'instruction DAA.

```
?R
 F=08  A=1D  C=00  B=00  E=45  D
=40 HL=411A IX=028F IY=4000

?
```

Contenu des registres après exécution de l'instruction DAA.

```
?R
 F=30  A=23  C=00  B=00  E=45  D
=40 HL=411A IX=028F IY=4000

?
```

DEC

DEC r

r est un registre, une position mémoire adressée par les registres HL, IX ou IY, ou une paire de registres.

L'instruction DEC r décrémente le contenu de l'opérande r.

Remarque : Si l'opérande r est une paire de registres, le registre IX ou le registre IY, les indicateurs ne seront pas affectés.

Exemple: Décrémentation de la paire de registre BC.

LD	BC, 0020 H	01	20	00
DEC	BC	0B		
RET		C9		

Contenu des registres après exécution de l'instruction DEC.

```
?R
 F=20  A=5F  C=1F  B=00  E=45  D
=40 HL=411A IX=028F IY=4000

?
```

DI

Interdiction des interruptions.

L'instruction DI interdit, dès qu'elle est exécutée, la prise en compte des interruptions masquables.

DJNZ

DJNZ dd

dd est un déplacement compris entre – 128 et + 127.

L'instruction DJNZ décrémente le contenu du registre B et effectue le saut relatif demandé si son contenu est non nul. Si le contenu du registre B après décrémentation est nul le saut relatif n'est pas effectué et l'instruction suivante est exécutée.

Exemple: Mise à zéro d'une table de 10 octets.

Le premier octet de la table se trouve à l'adresse décimale 16700.

	LD	HL, 16700	21	3C	41
	LD	B, 0A H	06	0A	
	LD	C, 05	0E	05	
SUITE	LD	(HL), 0	36	00	
	INC	HL	23		
	DJNZ	SUITE	10	FB	
	RET		C9		

Contenu des registres avant exécution de l'instruction RET. Vous pouvez remarquer que le contenu du registre B est devenu nul et que celui de la paire de registres HL a été incrémenté de dix.

```
?R
 F=00   A=67   C=05   B=00   E=45   D
=40  HL=4146  IX=028F  IY=4000

?
```

Remarque: Pour effectuer le déplacement relatif, le microprocesseur Z 80 ajoute au contenu du compteur ordinal la valeur du déplacement

spécifié dans l'instruction. N'oubliez pas qu'au moment d'effectuer le saut le contenu du compteur ordinal pointe déjà sur l'instruction qui se trouve juste après l'instruction DJNZ. Il faudra donc enlever 2 au déplacement ainsi l'instruction DJNZ suivante qui boucle sur elle-même,

	SUITE	DJNZ	SUITE
ne sera pas codée	SUITE	DJNZ	00
mais	SUITE	DJNZ	– 2

c'est-à-dire 10 FE en hexédécimal.

EI

Autorisation des interruptions.

L'instruction EI autorise dès qu'elle est exécutée la prise en compte des interruptions masquables.

EX

EX AF, AF'
EX DE, HL
EX (SP), HL ou IX ou IY

L'instruction EX permet d'échanger le contenu du premier opérande avec le contenu du second opérande.

Exemple: Échange des contenus des paires de registres HL et DE.

LD	HL, 1234 H	21	34	12
LD	DE, 5678 H	11	78	56
EX	DE, HL	EB		
RET		C9		

Contenu des registres après l'échange des paires de registres HL et DE.

```
?R
 F=08  A=61  C=00  B=00  E=34  D
=12 HL=5678 IX=028F IY=4000

?
```

EXX

L'instruction EXX permet d'échanger les contenus des registres primaires et des registres secondaires sauf les registres AF et AF' (Pour échanger les registres AF et AF' il faut utiliser l'instruction EX AF, AF').

HALT

L'instruction HALT arrête le fonctionnement de l'unité centrale. Cette dernière exécutera alors des NOP. L'unité centrale reprendra son fonctionnement normal sur la réception d'une interruption.

Remarque: N'exécutez surtout pas cette instruction si votre ZX 81 est dans le mode **FAST**.

IM

L'instruction IM permet de sélectionner l'un des modes d'interruptions du Z 80.

Dans le mode IM0 le circuit provoquant l'interruption placera sur le bus de données uné instruction à exécuter.

Dans le mode IM1 — celui qui est utilisé dans le ZX 81 — une interruption provoquera l'exécution de l'instruction RST 0038 H.

Dans le mode IM2 le contenu du registre I est utilisé comme partie haute d'une adresse, la partie basse de l'adresse étant fournie par le circuit qui provoque l'interruption. L'adresse ainsi obtenue pointe sur le vecteur d'interruption.

IN

IN r, (C)

r est un des registres A, B, C, D, E, H ou L.

L'instruction IN r, (C) permet de charger le registre r avec la valeur lue sur l'élément périphérique adressé par la paire de registres BC. Le

registre B donne les poids forts de l'adresse de l'élément périphérique et le registre C les poids faibles. Cette instruction modifie les indicateurs.

IN A, (N)

L'instruction IN A, (N) charge l'accumulateur avec la valeur lue sur l'élément périphérique N. Cette instruction ne modifie pas les indicateurs.

INC

INC r

r est un registre, une paire de registres ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction INC r incrémente de une unité le contenu de l'opérande spécifié.

Exemple: Incrémentation de la paire de registres HL.

LD	HL, 00FFH	21	FF	00
INC	HL	23		
RET		C9		

```
?R
 F=28  A=5F  C=00  B=00  E=45  D
=40 HL=0100 IX=028F IY=4000

?
```

IND

L'instruction IND charge la position mémoire adressée par la paire de registres HL avec la valeur lue sur le périphérique adressé par le contenu du registre C. Ceci est suivi d'une décrémentation du registre B et de la paire de registres HL de une unité.

INDR

Cette instruction est analogue à l'instruction IND. Toutefois l'instruction sera réexécutée tant que le contenu du registre B ne sera pas nul.

INI

Analogue à l'instruction IND ; la paire de registres HL est incrémentée au lieu d'être décrémenté.

INIR

Analogue à l'instruction INDR ; la paire de registres HL est incrémentée au lieu d'être décrémentée.

JP

JP cc, add

cc est une condition.
add est une adresse.

Débranchement conditionnel. Si la condition cc est vraie alors le saut à l'adresse indiquée est effectué. Sinon l'instruction suivante est exécutée.

JP add

add est une adresse.

Débranchement inconditionnel à l'adresse indiquée.

JP (r)

r peut être la paire de registres (HL), le registre IX ou le registre IY.

Le contenu du registre spécifié est pris comme une adresse, le débranchement est ensuite effectué à cette adresse.

Exemple: Ce programme initialise le registre B à FF si le contenu de l'accumulateur est nul. (Le programme commence à l'adresse 16675.)

	LD	A, 00	3E	00	
	LD	B, 0A H	06	0A	
	OR	A	B7		
	JP	NZ, APRES	C2	2D	41
	LD	B, FF H	06	FF	
APRES					
	RET		C9		

```
?R
 F=44   A=00   C=00   B=FF   E=45   D
=40 HL=411A IX=028F IY=4000

?
```

JR

JR cc, e

cc est une condition.
e est un déplacement relatif.

Si la condition est vraie, le saut relatif e est effectué. Sinon l'instruction suivante est exécutée.

JR e

Débranchement relatif de e.

Exemple: Reprenons l'exemple précédent en utilisant un débranchement relatif et changeons le contenu de l'accumulateur.

	LD	A, 10 H	3E	10		
	LD	B, 0A H	06	0A		
	OR	A	B7			
	JR	NZ, APRES	20	02		
	LD	B, FF H	06	FF		
APRES						
	RET		C9			

```
?R
 F=00   A=10   C=00   B=0A   E=45   D
=40 HL=411A IX=028F IY=4000
```

LD

LD r1, r2

r1 et r2 peuvent être une paire de registres, un registre, une adresse mémoire, une constante, etc... (Reportez-vous au livre, l'assembleur facile du Z 80 paru dans la même collection).

Exemple: Mise à zéro de l'emplacement mémoire 16700.

LD	HL, 16700	21	3C	41
LD	(HL), 00	36	00	
RET		C9		

LDD

L'instruction LDD charge l'emplacement mémoire adressé par la paire de registres DE avec le contenu de l'emplacement mémoire adressé par la paire de registres HL et décrémente de une unité les paires de registres HL, BC et DE.

LDDR

Cette instruction est analogue à l'instruction LDD. Toutefois l'instruction sera réexécutée jusqu'à ce que le contenu de la paire de registres BC soit nul.

LDI

Cette instruction est analogue à l'instruction LDD. Toutefois les contenus des paires de registres HL et DE sont incrémentés de une unité au lieu d'être décrémentés. Le contenu de la paire de registres BC est toujours décrémenté de une unité.

LDIR

Cette instruction est analogue à l'instruction LDI. L'instruction sera cependant réexécutée jusqu'à ce que le contenu de la paire de registres BC soit nul.

Exemple: Avec l'instruction LDI.

Le petit programme suivant recopie le contenu de l'adresse décimale 16705 à l'adresse décimale 16700. On initialisera auparavant le contenu de ces deux positions mémoire avec respectivement 00 et FF (255 en décimal).

16700	=	413C	en hexadécimal
16705	=	4141	en hexadécimal

LD	BC, 0005	01	05	00
LD	DE, 413C H	11	3C	41
LD	HL, 4141 H	21	41	41
LDI		ED	A0	
RET		C9		

Contenu des registres après l'exécution de l'instruction LDI.

```
?R
 F=04   A=65   C=04   B=00   E=3D   D
=41  HL=4142  IX=028F  IY=4000

?
```

Après exécution de l'instruction LDI, le contenu de la position mémoire 16700 est 00.

NEG

L'instruction NEG remplace le contenu de l'accumulateur par son complément à 2.

Exemple:

LD	A, 55H	3E	55	
NEG		ED	44	
RET		C9		

Contenu des registres après exécution de l'instruction NEG.

```
?R
 F=BB   A=AB   C=00   B=00   E=45   D
=40  HL=411A  IX=028F  IY=4000

?
```

Remarque: Voyez la différence avec l'instruction CPL.

NOP

L'instruction NOP ne fait ... rien! En fait elle est surtout utilisée pour supprimer des instructions en trop lors de correction d'un pro-

gramme ce qui vous évite ainsi de recalculer tous les débranchements ou pour perdre du temps lors de temporisations.

OR

OR r

r est un registre, une constante ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction OR réalise la fonction OU logique entre l'accumulateur et l'opérande r. Le résultat est rangé dans l'accumulateur.

Exemple: Mise à 1 des quatre bits de poids forts de l'accumulateur.

LD	A, 01	3E	01	
OR	F0 H	F6	F0	
RET		C9		

```
?R
 F=A0  A=F1  C=00  B=00  E=45  D
=40 HL=411A IX=026F IY=4000

?
```

Remarque: Comme l'instruction AND, cette instruction est souvent utilisée pour positionner les flags.

OUT

OUT (C), r

r est un des registres A, B, C, D, E, H ou L.

L'instruction OUT recopie le contenu du registre r dans l'élément périphérique adressé par la paire de registres BC. Le registre B donne les poids forts de l'adresse de l'élément périphérique et le registre C les poids faibles. Cette instruction ne modifie pas les indicateurs.

OUT (N), A

L'instruction OUT (N), A recopie le contenu de l'accumulateur dans l'élément périphérique N. Cette instruction ne modifie pas les indicateurs.

OUTD

L'instruction OUTD recopie le contenu de l'emplacement mémoire adressé par la paire de registres HL dans l'élément périphérique adressé par le contenu du registre C et décrémente les contenus de la paire de registres HL et du registre B de une unité.

OTDR

Cette instruction est analogue à l'instruction OUTD. L'instruction est réexécutée jusqu'à ce que le contenu du registre B soit nul.

OUTI

Cette instruction est analogue à l'instruction OUTD. Toutefois le contenu de la paire de registres HL est incrémenté au lieu d'être décrémenté.

OTIR

L'instruction OTIR est analogue à l'instruction OUTI. L'instruction est réexécutée jusqu'à ce que le contenu du registre B soit nul.

POP

POP rp

rp est une des paires de registres BC, DE, HL ou AF ou le registre IX ou IY.

L'instruction POP permet de dépiler le registre double rp. L'emplacement mémoire adressé par le pointeur de pile est stocké dans les poids faibles de la paire de registres spécifiée et l'emplacement mémoire suivant dans les poids forts. Le pointeur de pile est ensuite incrémenté de 2.

PUSH

PUSH rp

rp est une des paires de registres BC, DE, HL ou AF ou le registre IX ou IY.

L'instruction PUSH permet d'empiler le registre double rp. Le pointeur de pile est décrémenté et les poids forts de la paire de registres spécifiée sont stockés dans l'emplacement mémoire adressé par le pointeur de pile. Les poids faibles sont stockés dans l'emplacement précédent. Le pointeur de pile est de nouveau décrémenté.

Exemple: Échange des contenus des paires de registres BC et DE.

LD	DE, 1234 H	11	34	12
LD	BC, 5678 H	01	78	56
PUSH	DE	D5		
PUSH	BC	C5		
POP	DE	D1		
POP	BC	C1		
RET		C9		

```
?R
 F=20  A=65  C=34  B=12  E=78  D
=56 HL=411A IX=028F IY=4000
```

RES

RES b, r

r est un registre ou une position mémoire adressée par les registres HL, IX ou IY.

b est un nombre compris entre 0 et 7.

L'instruction RES b, r remet à zéro le bit b de l'opérande r.

Exemple: Mise à zéro du bit 4 de l'accumulateur.

LD	A, FF H	3E	FF	
RES	4, A	CB	A7	
RET		C9		

```
?R
 F=00  A=EF  C=00  B=00  E=45  D
=40 HL=411A IX=028F IY=4000
```

RET

RET cc

cc est une condition.

Retour conditionnel. Si la condition cc est vraie, le retour au programme appelant est effectué sinon l'instruction suivante est exécutée. L'adresse de retour doit être contenue dans les deux premiers octets de la pile.

RET

L'instruction RET provoque le retour au programme appelant. L'adresse de retour doit être contenue dans les deux premiers octets de la pile.

RETI

L'instruction RETI doit être la dernière instruction d'un sous-programme de traitement d'une interruption. Les deux premiers octets de la pile doivent contenir l'adresse de l'instruction qui aurait été exécutée si il n'y avait pas eu d'interruption.

RETN

L'instruction RETN doit être la dernière instruction du sous-programme de traitement de l'interruption non masquable.

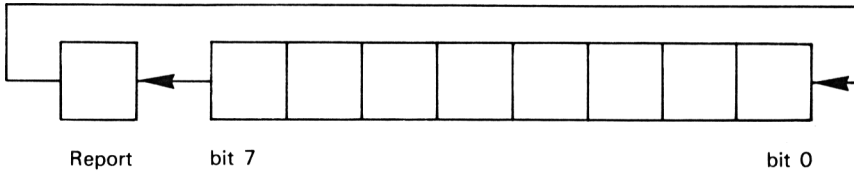
RL

RL r

r est un registre ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction RL r décale de un bit vers la gauche et à travers le report, le contenu de l'opérande r. Le contenu du report est recopié dans

le bit 0 et le bit 7 dans le report. Le résultat est stocké dans l'opérande r.



Exemple: Rotation du registre B. Le contenu du report est forcé à un avant le décalage.

LD	B, 55 H	06	55	
SCF		37		
RL	B	CB	10	
RET		C9		

```
?R
 F=A8  A=5F  C=00  B=AB  E=45  D
=40 HL=411A IX=028F IY=4000
```

RLA

L'instruction RLA décale le contenu de l'accumulateur de un bit vers la gauche et à travers le report. Le contenu du report est recopié dans le bit 0 et le bit 7 dans le report. Le résultat est stocké dans l'accumulateur.

Exemple:

LD	A, 55	3E	55	
SCF		37		
RLA		17		
RET		C9		

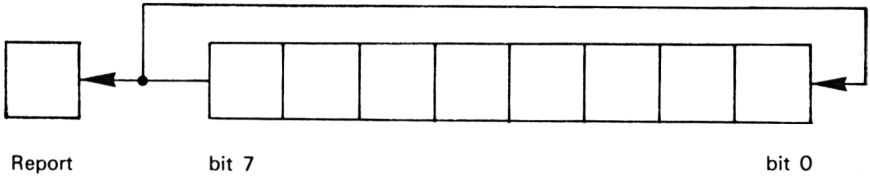
```
?R
 F=28  A=AB  C=00  B=00  E=45  D
=40 HL=411A IX=028F IY=4000

?
```

RLC

RLC r

r est un des registres ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction RLC r décale le contenu du registre r de un bit vers la gauche sans le report. Le bit 7 est recopié à la fois dans le report et dans le bit 0. Le résultat est stocké dans l'opérande r.



Exemple:

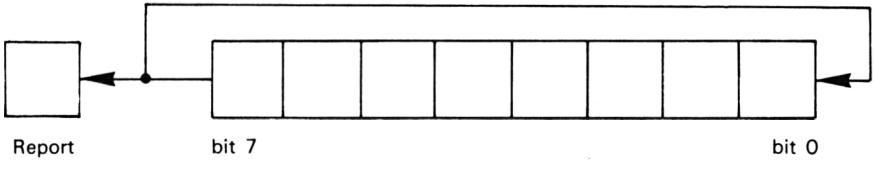
Rotation du registre B.

LD	B, 55 H	06	55	
SCF		37		
RLC	B	CB	00	
RET		C9		

```
?R
 F=AC   A=5F   C=00   B=AA   E=45   D
=40 HL=411A IX=028F IY=4000
```

RLCA

L'instruction RLCA décale le contenu de l'accumulateur de un bit vers la gauche sans le report. Le bit 7 est recopié à la fois dans le report et dans le bit 0. Le résultat est stocké dans l'accumulateur.



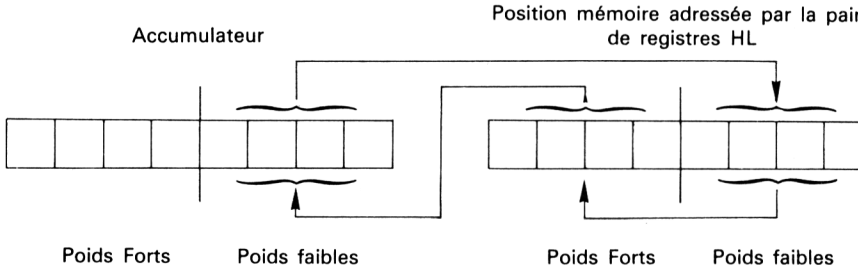
Exemple:

LD	A, 55 H	3E	55	
SCF		37		
RLCA		07		
RET		C9		

```
?R
 F=28   A=AA   C=00   B=00   E=45   D
=40  HL=411A  IX=028F  IY=4000

?
```

RLD

L'instruction RLD recopie les quatre bits de poids faibles de la position mémoire adressée par la paire de registres HL dans les quatre bits de poids forts de cette même position mémoire. Les quatre bits de poids forts sont recopiés dans les quatre bits de poids faibles de l'accumulateur ces derniers étant eux-mêmes recopiés dans les quatre bits de poids faibles de la position mémoire précédente (c'est-à-dire celle adressée par la paire de registres HL).



Exemple: L'emplacement mémoire utilisé est celui situé à l'adresse décimale 16700.

LD	A, 01	3E	01	
LD	HL, 16700	21	3C	41
LD	(HL), 23 H	36	23	
RLD		ED	6F	
RET		C9		

```
?R
 F=00   A=02   C=00   B=00   E=45   D
=40 HL=413C  IX=028F  IY=4000
```

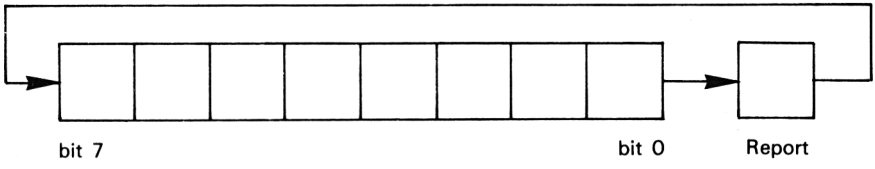
Après exécution de l'instruction RLD le contenu de la position mémoire 16700 est 31 H.

RR

RR r

r est un registre ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction RR r décale le contenu de l'opérande r de un bit vers la droite à travers le report. Le bit 0 est recopié dans le report et le report dans le bit 7. Le résultat est stocké dans l'opérande r.



Exemple: Rotation du registre B. Le contenu du report est forcé à zéro.

LD	B, 55 H	06	55
AND	A	A7	
RR	B	CB	18
RET		C9	

```
?R
 F=29   A=5F   C=00   B=2A   E=45   D
=40 HL=411A  IX=028F  IY=4000

?
```

RRA

L'instruction RRA décale le contenu de l'accumulateur de un bit vers la droite à travers le report. Le résultat est stocké dans l'accumulateur.

RRC

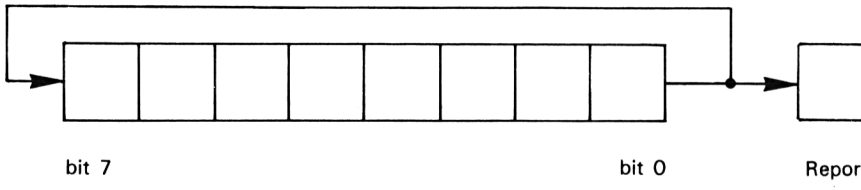
r est un registre ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction RRC décale le contenu de l'opérande r de un bit vers la droite sans le report. Le bit 0 est recopié à la fois dans le bit 7 et dans le report. Le résultat est stocké dans l'opérande r.

Exemple: Rotation du registre B.

LD	B, 55 H	06	55		
AND	A	A7			
RRC	B	CB	08		
RET		C9			

```
?R
 F=AD   A=5F   C=00   B=AA   E=45   D
=40  HL=411A  IX=028F  IY=4000

?
```

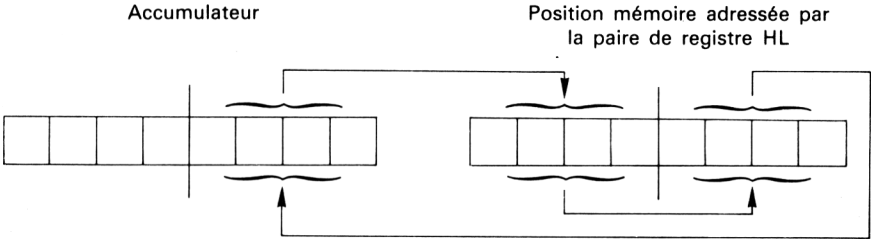


RRCA

L'instruction RRCA décale le contenu de l'accumulateur de un bit vers la droite sans le report. Le résultat est stocké dans l'accumulateur.

RRD

L'instruction RRD recopie les quatre bits de poids forts de la position mémoire adressée par la paire de registres HL dans les quatre bits de poids faibles de cette même position mémoire. Les quatre bits de poids faibles sont recopiés dans les quatre bits de poids faibles de l'accumulateur qui sont eux-mêmes recopiés dans les quatre bits de poids forts de la position mémoire précédente.



Exemple :

LD	A, 01	3E	01	
LD	HL, 16700	21	3C	41
LD	(HL), 23 H	36	23	
RRD		ED	67	
RET		C9		

```
?R
 F=04   A=03   C=00   B=00   E=45   D
=40  HL=413C  IX=028F  IY=4000
```

Avant exécution de l'instruction RRD, le contenu de la position mémoire 16700 était 23 H. Après exécution de l'instruction RRD ce contenu est devenu 12 H

RST

RST add

add est une adresse privilégiée et peut prendre les valeurs hexadécimales 00 H, 08 H, 10 H, 18 H, 20 H, 28 H, 30 H, 38 H.

L'exécution de l'instruction RST x provoque la mémorisation dans la pile de l'adresse de l'instruction suivante et le débranchement à l'adresse add.

SBC

SBC A, r

r est un registre, une constante ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction SBC A, r soustrait du contenu de l'accumulateur les contenus de l'opérande r et de l'indicateur de report. Le résultat est stocké dans l'accumulateur.

Exemple: (l'indicateur de report est au préalable forcé à 1).

LD	A, 15	3E	0F	
SCF		37		
SBC	A, 4	DE	04	
RET		C9		

```
?R
 F=0A   A=0A   C=00   B=00   E=45   D
=40  HL=411A  IX=028F  IY=4000

?
```

SBC HL, rp

rp est une paire de registres (BC, DE, HL ou SP).

L'instruction SBC HL, rp soustrait du contenu de la paire de registres HL les contenus de la paire de registres rp et de l'indicateur de report. Le résultat est rangé dans la paire de registres HL.

Exemple: (L'indicateur de report est au préalable forcé à 1).

LD	HL, 1111 H	21	11	11
LD	BC, 0010 H	01	10	00
SCF		37		
SBC	HL, BC	ED	42	
RET		C9		

```
?R
 F=02   A=63   C=10   B=00   E=45   D
=40  HL=1100  IX=028F  IY=4000

?
```

SCF

L'instruction SCF met à un l'indicateur de report.

SET

SET b, r

r est un registre ou une position mémoire adressée par les registres HL, IX ou IY.

b est un nombre compris entre 0 et 7

L'instruction SET b, r met à 1 le bit b de l'opérande r.

Exemple: Mise à 1 du bit 4 de l'accumulateur.

LD	A, 03	3E	03	
SET	4, A	CB	E7	
RET		C9		

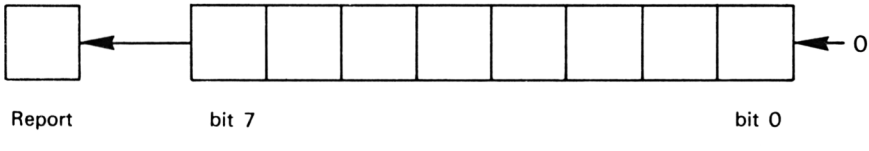
```
?R
 F=20  A=13  C=00  B=00  E=45  D
=40 HL=411A IX=028F IY=4000
```

SLA

SLA r

r est un registre ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction SLA décale *arithmétiquement* vers la gauche le contenu de l'opérande R. Le bit 7 est recopié dans le report et le bit 0 est forcé à zéro.



Exemple:

LD	A, A3 H	3E	A3
SLA	A	CB	27
RET		C9	

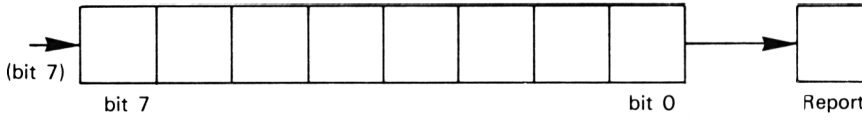
```
?R
 F=01  A=46  C=00  B=00  E=45  D
=40 HL=411A IX=028F IY=4000
```

SRA

SRA r

r est un registre ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction SRA décale *arithmétiquement* vers la droite le contenu de l'opérande r. Le bit 0 est recopié dans le report et le bit 7 ne change pas.



LD	A, A4 H	3E	A4	
SRA	A	CB	2F	
RET		C9		

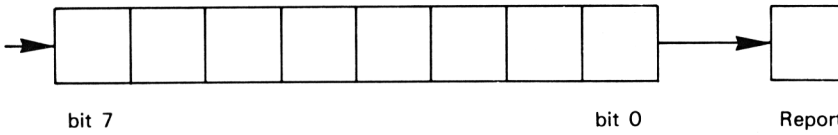
```
?R
 F=84  A=D2  C=00  B=00  E=45  D
=40 HL=411A IX=028F IY=4000 SP=7
FEA
```

SRL

SRL r

r est un registre ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction SRL décale *logiquement* vers la droite le contenu de l'opérande r. Le bit 0 est recopié dans le report et le bit 7 est forcé à zéro.



Exemple:

LD	A, 81 H	3E	81	
SRL	A	CB	3F	
RET		C9		

```
?R
 F=01   A=40   C=00   B=00   E=45   D
=40  HL=411A  IX=028F  IY=4000
```

SUB

SUB A, r

r est un registre, une constante ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction SUB soustrait du contenu de l'accumulateur, le contenu de l'opérande r. Le résultat est stocké dans l'accumulateur.

Exemple:

LD	A, 15	3E	0F	
SCF		37		
SUB	A, 4	D6	04	
RET		C9		

```
?R
 F=0A   A=0B   C=00   B=00   E=45   D
=40  HL=411A  IX=028F  IY=4000

?
```

XOR

XOR r

r est un registre, une constante ou une position mémoire adressée par les registres HL, IX ou IY.

L'instruction XOR réalise une fonction OU exclusif entre le contenu de l'accumulateur et le contenu de l'opérande r. Le résultat est rangé dans l'accumulateur.

Exemple :

LD	A, 55 H	3E	55	
XOR	0F H	EE	0F	
RET		C9		

```
?R
 F=0C   A=5A   C=00   B=00   E=45   D
=40  HL=411A  IX=028F  IY=4000

?
```

Remarque : Cette instruction est parfois utilisée pour remettre l'accumulateur à zéro.

Exemple :

XOR	A	AF		
RET		C9		

```
?R
 F=44   A=00   C=00   B=00   E=45   D
=40  HL=411A  IX=028F  IY=4000

?
```

2

Où et comment stocker un programme écrit en langage machine

Après avoir revu les instructions du Z 80, voyons maintenant comment rentrer un programme en langage machine.

Dans ce chapitre nous reprenons les trois méthodes, proposées dans le manuel SINCLAIR, permettant de stocker un programme écrit en langage machine. Nous verrons ensuite comment générer *rapidement et simplement* une instruction REM de plusieurs K octets et un moyen de sauvegarder un programme en langage machine situé dans le haut de la mémoire.

2.1. MÉTHODES POUR STOCKER UN PROGRAMME ÉCRIT EN LANGAGE MACHINE

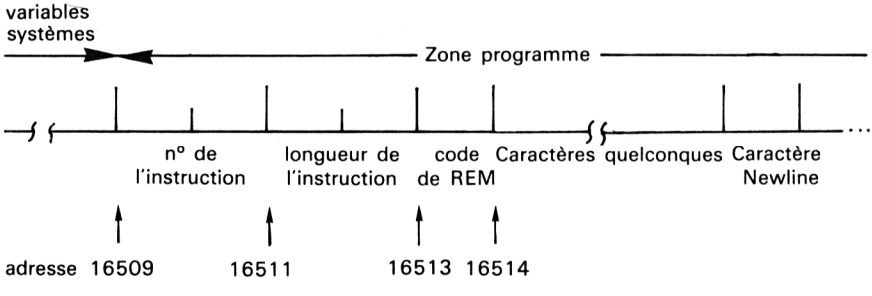
Les trois méthodes proposées par SINCLAIR reposent sur deux points essentiels.

1. Réserver une zone mémoire de taille suffisante pour contenir le programme en langage machine de manière telle qu'il ne soit pas "détruit" par le BASIC.

2. Connaître toujours de façon exacte l'adresse à laquelle cette zone mémoire commence.

2.1.1. Utilisation de l'instruction REM

La réservation de la taille mémoire désirée se fera avec une instruction REM comportant un nombre de caractères égal au nombre d'octets du programme écrit en langage machine. Le programme en langage machine sera alors chargé dans cette instruction REM par des instructions POKE. Cette instruction REM peut se trouver n'importe où dans votre programme BASIC toutefois le calcul de l'adresse de début du sous-programme écrit en langage machine sera alors relativement délicat. En plaçant cette instruction REM en tête du programme, l'adresse de début du sous-programme écrit en langage machine sera toujours 16514. En effet la zone programme commence juste après la zone des variables systèmes, (reportez-vous au chapitre 27 du manuel SINCLAIR) la première instruction du programme commence donc à l'adresse 16509. Les cinq premiers octets de l'instruction BASIC servant à mémoriser son numéro de ligne, sa longueur et son code, l'adresse du premier octet disponible pour le sous-programme écrit en langage machine est donc 16514.



Exemple:

Le programme suivant trace une diagonale sur l'écran:

						Adresses
	LD	DE, 0034	11	22	00	16514
	LD	HL, (16396)	21	0C	40	16517
	INC	HL	23			16520
	LD	B, 20	06	14		16521
SUITE	LD	(HL), 86 H	36	86		16523
	ADD	HL, DE	19			16525
	DJNZ	SUITE	10	FB		16526
	RET		C9			16528

Nous le chargerons avec le programme BASIC suivant:

```
  10 REM XXXXXXXXXXXXXXX
  20 REM *** CHARGEMENT DU PROGRA
MME ***
  30 POKE 16514,17
  32 POKE 16515,34
  34 POKE 16516,0
  36 POKE 16517,42
  38 POKE 16518,12
  40 POKE 16519,64
  42 POKE 16520,35
  44 POKE 16521,6
  46 POKE 16522,20
  48 POKE 16523,54
  50 POKE 16524,134
  52 POKE 16525,25
  54 POKE 16526,16
  56 POKE 16527,251
  58 POKE 16528,201
  60 REM *** EXECUTION DU PROGRA
MME ***
  70 RAND USR 16514
  80 STOP
```

On peut aussi pour charger le programme utiliser le petit programme de chargement proposé au chapitre 3 avec lequel vous pouvez entrer directement le code hexadécimal alors que en utilisant l'instruction POKE vous devez reconvertir le code hexadécimal en décimal.

Remarque: Dès que le programme a été chargé dans l'instruction REM les instructions 30 à 58 peuvent être supprimées, le programme en langage machine ne pouvant plus être détruit par le BASIC (sauf, bien sûr, par des commandes POKE ou NEW).

Pourquoi utiliser une instruction REM pour réserver la place en mémoire et non une instruction GOTO ou PRINT ? En effet nous pourrions par exemple débuter notre programme par l'instruction.

```
10  PRINT xxxx ← 15 fois → xxx
```

Ceci réserve aussi 15 octets et il est tout à fait possible de venir y charger le programme en langage machine. Cependant quand nous lancerons l'exécution votre ZX 81 "reconnaîtra" l'instruction PRINT et il cherchera à éditer la valeur de cette variable au nom bizarre ! Cela pourrait être cause de quelques ennuis. L'avantage de l'instruction REM est qu'elle sert normalement à introduire un commentaire; aussi quand votre ZX 81 rencontre cette instruction il sait qu'il n'a rien à faire si ce n'est aller exécuter l'instruction BASIC suivante. Il ne cherchera donc pas à interpréter le code du programme en langage machine.

Remarque importante :

L'utilisation d'une instruction REM pour stocker un programme en langage machine est très fréquente. Il existe cependant un inconvénient auquel il faudra faire particulièrement attention. Certains codes particuliers peuvent être cause de problèmes lors d'une mise à jour du programme. C'est le cas du code hexadécimal 7E qui est le code utilisé pour indiquer que les cinq octets suivants mémorisent un nombre en virgule flottante pour l'interpréteur BASIC. Pour le Z 80 le code 7E est l'instruction LD A, (HL). Prenons un exemple.

Ce petit programme vient charger dans l'accumulateur le contenu de la variable système FLAGS et initialise le registre B à une certaine valeur (rien de très utile !)

LD	HL, 16385	21	01	40
LD	A, (HL)	7E		
LD	B, 04	06	04	
INC	B	04		
INC	B	04		
INC	B	04		
RET		C9		

Ce programme faisant 10 octets nous les réservons avec une instruction REM de 10 caractères :

```
10 REM XXXXXXXXXX
```

Rentrons le code du programme en langage machine par des commandes POKE. Le programme BASIC suivant vous permettra de vérifier que votre code est bien rentré.

```
10 REM 5■RNDTAN
20 FOR I=16514 TO 16523
25 LPRINT I;" ";PEEK I
30 NEXT I
40 STOP
```

Vous devez obtenir le résultat suivant :

```
16514 33
16515 1
16516 64
16517 126
16518 6
16519 4
16520 4
16521 4
16522 4
16523 201
```

Maintenant supposons que nous désirions rajouter une initialisation du registre C à ce programme par exemple :

```
LD  C, 2          0E  02
```

notre programme comportera donc deux octets de plus, nous ajouterons donc deux caractères x au bout de l'instruction 10 en utilisant la commande EDIT. Le listing sera le suivant :

```
10 REM 5* RNDTAN XX
20 FOR I=16514 TO 16523
25 LPRINT I;" ";PEEK I
30 NEXT I
40 STOP
```

Nous pouvons donc nous attendre à trouver aux adresses 16524 et 16525 le code du caractère x c'est-à-dire 61. Relançons l'exécution du programme de vérification en faisant terminer la boucle d'impression à l'adresse 16525. Le résultat est :

```
16514 33
16515 1
16516 64
16517 201
16518 61
16519 61
16520 118
16521 0
16522 20
16523 27
16524 0
16525 235
```

Ce qui ne correspond absolument pas au résultat attendu ! De notre programme il ne reste plus que les trois premiers octets (33, 1, 64) le dernier (201) et les deux caractères qui ont été rajoutés (61, 61). Ensuite

nous avons le caractère NEWLINE (118) puis le début de l'instruction 20, à savoir son numéro (0, 20), sa longueur 27 octets (27, 0) et le code de l'instruction FOR (235)*. Dans la manipulation 6 octets ont disparus ! Les six octets qui étaient censés mémoriser un chiffre en virgule flottante.

Ainsi si vous modifiez un programme en langage machine, contenu dans une instruction REM assurezvous que l'instruction LD A, (HL) n'a pas été utilisée. Si c'est le cas évitez l'emploi de la commande EDIT ou alors vous seriez obligé de réentrer le programme en entier.

Comment procéder alors ?

* Si vous avez la possibilité d'ajouter une nouvelle instruction REM juste après celle que vous désirez corriger, faites-le et reportez-vous au paragraphe 2.2.1. de ce chapitre. (Cette instruction REM pourra comporter six caractères de moins que le nombre d'octets que vous désirez insérer dans votre programme écrit en langage machine.)
* Vous n'avez pas la possibilité d'ajouter une nouvelle instruction REM juste après celle que vous désirez corriger. Procédez alors comme suit :

— Inclure dans votre programme BASIC une instruction REM ayant six caractères de plus que le nombre d'octets que vous désirez insérer dans votre programme écrit en langage machine.

— Calculer l'adresse de cette nouvelle instruction REM en utilisant le petit programme suivant (lancez son exécution par la commande GOTO 9500).

```
9500 LET W=16509
9505 LET C=256
9507 LET F=PEEK 16396+C*PEEK 163
97
9510 PRINT "NUMERO DE LIGNE ?";
9515 INPUT NL
9517 PRINT NL
9520 LET A=PEEK (W+1)+C*PEEK W
9525 LET B=4+PEEK (W+2)+C*PEEK (
W+3)
9530 IF A<>NL THEN GOTO 9550
9535 LET W=W+5
9540 PRINT "LE PREMIER OCTET DIS
PONIBLE DE L INSTRUCTION ";NL;"
SE TROUVE A L ADRESSE ";W
9545 STOP
9550 LET W=W+B
9555 IF W<F THEN GOTO 9520
9560 PRINT "L INSTRUCTION ";NL;"
 N EXISTE PAS"
```

* Voir le chapitre 2 de "*La conduite du ZX 81*" paru dans la même collection.

— Remplacer dans le programme en langage machine à corriger trois octets par l'instruction : JP adr. de la nouvelle instruction REM. (Dans notre exemple nous pourrions remplacer les deux instructions LD A, (HL) et LD B, 04 par JP adresse ...).

— Dans la nouvelle instruction REM, remettez les codes des trois octets qui ont été écrasés par l'instruction JP adresse. (Pour notre exemple ce serait les codes des instructions LD A, (HL) et LD B, 04), puis introduisez la modification désirée (LD C, 2 pour notre exemple) et terminez celle ci par un débranchement inconditionnel au traitement suivant du programme à corriger. (Pour notre exemple ce serait la première instruction INC B nous terminerions donc notre modification par l'instruction JP à l'adresse 16520).

2.1.2. Modification de la variable système RAMTOP

La variable système RAMTOP contient l'adresse du dernier octet de la mémoire plus un. Une commande NEW n'efface la mémoire que jusqu'à l'adresse mémorisée dans la variable système RAMTOP. Ainsi la modification par des instructions POKE du contenu de cette variable suivie de l'exécution de la commande NEW laissera dans le haut de la mémoire une place disponible utilisable pour stocker un sous-programme écrit en langage machine. Ce sous-programme sera ainsi à l'abri d'une commande NEW, ce qui n'est pas le cas pour l'instruction REM. Vous pouvez alors venir charger votre programme par des instructions POKE.

Avec cette méthode nous pouvons réserver un grand nombre d'octets rapidement. En effet supposons que nous désirions réserver une zone de 768 octets. Avec un bloc mémoire 16 K, ceci reviendra à forcer le contenu de la variable RAMTOP à 32000 (le dernier octet disponible avec un bloc mémoire 16 K occupe l'adresse 32767) en procédant ainsi :

```
POKE   16388, 0
POKE   16389, 125
NEW
```

Vous disposez maintenant d'une zone de 768 octets que vous pouvez venir " remplir " par des POKE. L'adresse de début de cette zone est :

```
256 * PEEK 16389 + PEEK 16388
```

Cela est évidemment très pratique (imaginez-vous en train de rentrer une instruction REM de 768 caractères !...). Cependant avec cette méthode votre programme écrit en langage machine ne pourra pas être sauvegardé, la commande SAVE ignorant tout ce qui se trouve au-dessus de l'adresse pointée par la variable système RAMTOP. (Nous verrons comment remédier à cet inconvénient plus loin).

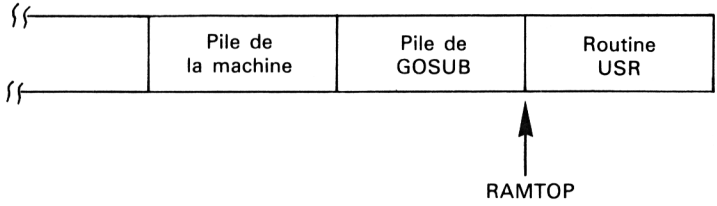
Reprenons l'exemple précédent le programme sera :

```
10 REM CALCUL ADRESSE DE RAMTO
P
20 LET A=PEEK 16388+256*PEEK 1
6389
30 REM CHARGEMENT DU PROGRAMME
32 POKE A,17
34 POKE A+1,34
36 POKE A+2,0
37 POKE A+3,42
38 POKE A+4,12
40 POKE A+5,64
42 POKE A+6,35
44 POKE A+7,6
46 POKE A+8,20
48 POKE A+9,54
50 POKE A+10,134
52 POKE A+11,25
54 POKE A+12,16
56 POKE A+13,251
58 POKE A+14,201
60 REM EXECUTION DU PROGRAMME
70 RAND USR A
80 STOP
```

Remarque : Les instructions 30 à 58 pourront être supprimées dès que le programme aura été chargé dans le haut de la mémoire.

Pourquoi faut-il entrer la commande NEW ?

Pour avoir une réponse satisfaisante reportons-nous au schéma de l'organisation de la mémoire donné dans le manuel SINCLAIR p. 171 et intéressons-nous au haut de la mémoire.



Nous voyons que dans le haut de la mémoire le ZX 81 initialise une pile. Lorsque la commande NEW est exécutée votre ZX 81 ira rechercher la nouvelle adresse de RAMTOP et réinitialisera sa pile juste avant.

Si vous n'exécutiez pas la commande NEW, en rentrant votre programme en langage machine, vous viendriez écraser le contenu de cette pile ce qui rendrait le fonctionnement de votre ZX 81 anormal. Nous pouvons vérifier ceci très simplement. Pour cela débrancher et rebrancher votre ZX 81. La pile du ZX 81 commencera à l'adresse 32765 (si vous travaillez avec un bloc de 16 K RAM).

Entrez la commande POKE 32765,34 (POKE 17405,34 si vous n'avez pas de bloc extension mémoire). Rien n'apparaît sur l'écran, et aucun appui touche n'est pris en considération. Le ZX 81 est "planté". Que s'est-il passé : lorsque vous avez entré votre commande le ZX 81 a sauvegardé dans sa pile certaines adresses ; la commande POKE 32765 vient détruire l'une de ces adresses ce qui lui donne finalement ce fonctionnement incohérent.

En conclusion la commande NEW est nécessaire pour que le ZX 81 puisse réinitialiser les piles qu'il gère dans le haut de la mémoire.

2.1.3. Utilisation d'une variable ou d'un tableau alphanumérique

En déclarant en tête de votre programme une variable alphanumérique comme suit :

```
5 LET C$ = "XX ........ XX"
           n caractères
```

ou un tableau alphanumérique avec l'instruction DIM :

5 DIM C$ (n) n = nombre d'octets du sous-programme en langage machine.

Vous réserverez une zone mémoire située au début de la zone des variables (juste après celle du fichier d'affichage).

Cette zone pourra alors être initialisée par des POKE ou à l'aide de la fonction CHR$. L'adresse de début de cette zone sera déterminée par :

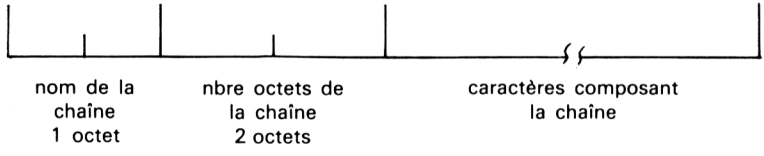
10 LET A = 3 + PEEK 16400 + 256 * PEEK 16401

pour la variable alphanumérique :

et 10 LET A = 6 + PEEK 16400 + 256 * PEEK 16401

pour le tableau alphanumérique.

en effet une chaîne de caractère est mémorisée comme suit :



le premier octet disponible de la chaîne de caractères est donc obtenu en ajoutant 3 à l'adresse de la chaîne.

De même pour le tableau alphanumérique il faudra ajouter 6 (reportez-vous au schéma p. 174 du manuel SINCLAIR).

Pour charger le programme nous pouvons réutiliser le programme proposé pour la modification de la variable RAMTOP, en modifiant l'instruction 10 comme précisé ci-avant. Il est aussi possible d'utiliser la fonction CHR$ au lieu de l'instruction POKE.

```
  5 DIM B$(15)
 10 LET A=6+PEEK 16400+256*PEEK
16401
 12 REM CHARGEMET DU PROGRAMME
 14 LET B$(1)=CHR$ 17
 16 LET B$(2)=CHR$ 34
 18 LET B$(3)=CHR$ 0
 20 LET B$(4)=CHR$ 42
 22 LET B$(5)=CHR$ 12
 24 LET B$(6)=CHR$ 64
 26 LET B$(7)=CHR$ 35
 28 LET B$(8)=CHR$ 6
 30 LET B$(9)=CHR$ 20
 32 LET B$(10)=CHR$ 54
 34 LET B$(11)=CHR$ 134
 36 LET B$(12)=CHR$ 25
 38 LET B$(13)=CHR$ 16
 40 LET B$(14)=CHR$ 251
 42 LET B$(15)=CHR$ 201
 46 REM EXECUTION DU PROGRAMME
 50 RAND USR A
 60 STOP
```

Remarque: Dès que le programme aura été chargé les instructions 12 à 42 pourront être supprimées.

Avec cette méthode il ne faudra plus utiliser la commande RUN mais GOTO (en effet RUN remet toute la zone variable à zéro, ce qui effacerait votre programme en langage machine). De plus la zone des variables se situant juste après le buffer d'affichage, elle changera de place suivant l'état de celui-ci, principalement lorsque l'on travaille sans le bloc 16 K. Vérifions ce dernier point.

Après avoir défiché votre bloc mémoire RAM 16 K. Entrer le petit programme suivant:

```
 10 DIM C$(200)
 20 LET C$(1)=CHR$ 201
 30 LET C$(3)=CHR$ 195
 40 LET C$(6)=CHR$ 201
 45 PRINT "ZX"
 50 LET A=6+PEEK 16400+256*PEEK
16401
 60 RAND USR A
 70 STOP
```

La première instruction du programme en langage machine est une instruction RET. Les instructions suivantes ne seront donc pas exécutées. Lancez l'exécution de ce programme. Tout se passe normale-

ment "ZX" est affiché et vous obtenez le report 9/70 dans le bas de l'écran.

Rajoutez maintenant l'instruction 52 CLS et relancez l'exécution du programme. Il se "plante"!...

Ceci est du au fait que l'instruction CLS a reconfiguré le buffer d'affichage réalisant ainsi un déplacement de la zone des variables (qui rappelez-vous se trouve juste après le buffer d'affichage) ce qui a modifié l'adresse de début de cette zone, c'est-à-dire l'adresse de début du sous-programme écrit en langage machine.

Avec le bloc extension 16 K RAM le buffer d'affichage ayant une taille fixe il n'y aura aucune différence de fonctionnement entre les deux cas.

Ainsi parmi les trois méthodes proposées, nous en retiendrons deux: l'utilisation d'une instruction REM et la modification de la variable système RAMTOP. Toutefois ces deux méthodes possèdent les inconvénients suivants:

— il est long et fastidieux de réserver une instruction REM de plusieurs centaines de caractères

— un programme en langage machine stocké dans le haut de la mémoire ne pourra pas être sauvegardé sur cassette. Voyons comment remédier à ces deux inconvénients.

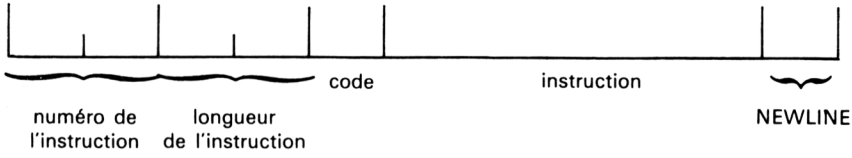
2.2. GÉNÉRATION D'UNE INSTRUCTION REM DE PLUSIEURS CENTAINES D'OCTETS

2.2.1. Avec deux instructions REM consécutives

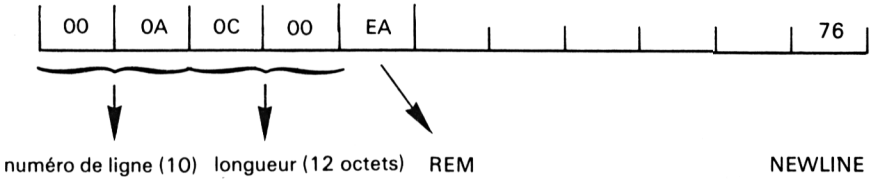
Rentrez deux instructions REM de 10 caractères chacune.

```
10  REM XXXXXXXXXX
20  REM XXXXXXXXXX
```

Pour chaque instruction, le ZX 81 mémorise en plus du code de l'instruction elle-même, sa longueur et son numéro de ligne. Une instruction sera stockée dans la zone programme de la manière suivante :



En réalité la longueur de l'instruction ne tient pas compte des quatre octets occupés par le numéro et la longueur elle-même. Ainsi l'instruction 10 REM X.....X aura, en mémoire, l'aspect suivant :



De même pour l'instruction :

```
20  REM X.....X
```

En modifiant la longueur de l'instruction REM nous pourrons condenser les deux instructions 10 et 20 en une seule ayant le numéro de ligne 10. Il suffit pour cela d'ajouter à la longueur de la première instruction REM, la longueur *réelle* de la seconde instruction REM (ie 16 = 12 + 4).

Pour le vérifier sur l'exemple entrez la commande :

```
POKE     16511, 28
```

28 = 12 (longueur de l'instruction 10) + 16 (longueur réelle de l'instruction 20).

La première instruction REM a sa longueur rangée aux adresses 16511 et 16512 (si elle est la première instruction du programme). En

venant ajouter à cette longueur, la longueur de la seconde instruction REM plus quatre (pour tenir compte des deux octets mémorisant les numéros de la ligne et des deux octets mémorisant la longueur) nous obtiendrons finalement une seule instruction REM.

```
POKE 16511, 28          (28 = 12 + 12 + 4)
```

Si vous rentrez maintenant l'instruction :

```
15  REM HHH
```

elle apparaîtra après "les" instructions 10 et 20.

Remarque :Vous pouvez être troublés par le fait que lorsque vous demandez un listage de votre programme "les" instructions 10 et 20 apparaissent sur deux lignes différentes. Pour remédier à ce petit inconvénient entrez la commande :

```
POKE 16524, 0
```

Vous écraserez ainsi le caractère "Newline" qui se trouve au bout de la ligne 10.

2.2.2. Utilisation d'un programme en langage machine

Imaginons un peu le travail que fait le ZX 81 quand vous lui demandez d'insérer une nouvelle ligne de programme syntaxiquement correcte. D'abord il faut qu'il recherche l'endroit exact où il doit insérer cette ligne. Ensuite il déplace toutes les autres instructions du programme de manière à se réserver une place suffisante pour venir y " loger" la nouvelle instruction. Ce déplacement implique bien évidemment une remise à jour des pointeurs. (Buffer d'affichage, Zone des variables, etc...).Ce travail fait, il ne reste plus qu'à ranger l'instruction et à lister le programme.

Il doit donc exister dans la ROM du ZX 81 un sous-programme qui permet de déplacer toute une partie de la mémoire en réalisant en même temps une mise à jour des variables système. Il existe en effet un programme permettant de décaler toute la mémoire d'un caractère. Ce programme se situe à l'adresse hexadécimale 0526 H. Appelons-le ESPACE.

Ce sous-programme nécessite dans la paire de registres HL l'adresse mémoire à laquelle on désire insérer le caractère et dans l'accumulateur le code du caractère à insérer.

Essayons le programme ci-dessous :

LD	HL, 16530	21	92	40
LD	A, 61	3E	3D	
CALL	ESPACE	CD	26	05
RET		C9		

Pour cela réservez une instruction REM de neuf caractères en tête de votre programme et utilisez la commande POKE ou le petit programme de chargement donné au chapitre 3 en modifiant l'instruction 9000 commme suit :

```
9000 FOR N=16514 TO 16522
```

Rajoutez l'instruction 20 REM AA votre listing sera donc :

```
10 REM 5RNDYXLN A TAN
20 REM AA
```

Lançons l'exécution du programme en langage machine par la commande RAND USR 16514 et demandons un listing.

```
10 REM 5RNDYXLN A TAN
20 REM AXA
```

Nous sommes venus insérer le caractère 'X' entre les deux 'A' de l'instruction 20 REM AA ce qui est bien ce que nous voulions faire. En effet 16530 est l'adresse du deuxième ' A ' de l'instruction 20 REM AA. Toutefois il ne faudra pas oublier de modifier par une commande POKE la longueur de l'instruction 20 sinon l'affichage risque de devenir incohérent ce qui peut créer quelques problèmes !...

Modifions donc notre programme pour qu'il mette à jour la longueur de l'instruction, et faisons-lui insérer 256 caractères.

DEBUT	LD	HL, 16546	21	A2	40
	LD	B, 0	06	00	
ENCORE	PUSH	BC	C5		
	PUSH	HL	E5		
	LD	A, 61	3E	3D	
	CALL	ESPACE	CD	26	05
	POP	HL	E1		
	POP	BC	C1		
	INC	HL	23		
	DJNZ	ENCORE	10	F4	
	LD	A, (16543)	3A	9F	40
	INC	A	3C		
	LD	(16543), A	32	9F	40
	RET		C9		

Comme précédemment nous utiliserons deux instructions REM la première contenant 25 caractères.

La sauvegarde des paires de registres BC et HL est nécessaire car leur contenu est détruit par le sous-programme ESPACE. L'adresse 16546 est l'adresse du deuxième A de l'instruction 20 REM AA. Nous insérerons donc les 256 caractères ' X ' entre les deux ' A ' de l'instruction 20. L'adresse 16543 contient l'octet de poids forts de la longueur de l'instruction. Puisque nous ajoutons 256 octets nous incrémentons son contenu de 1.

```
  10 REM 5[illegible]RND[illegible] VAL FAST YXLN A[illegible]
 LPRINT AT 7( POKE U[illegible]RNDWM[illegible]RNDTA
N
  20 REM AA
  30 RAND USR 16514
```

Après avoir fait une sauvergarde sur cassette et être passé en mode FAST, lancez l'exécution du programme par RUN. Le report C/F677 s'affiche dans le bas de l'écran. LIST vous donnera le résultat suivant :

```
  10 REM 5RND VAL FAST YXLN A
 LPRINT AT 7( POKE URNDWMRNDTA
N
  20 REM AXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXA
  30 RAND USR 16514
```

Remarque: Après avoir lancé une première fois l'exécution du programme ajoutez une nouvelle instruction entre les instructions 20 et 30 par exemple:

```
21  REM HH
```

puis enlevez-la en tapant 21 Newline.

Tout se passe très bien, de même si vous recommencez ceci après avoir relancé le programme. Par contre après avoir lancé trois fois le programme (l'instruction 20 a donc une longueur de 772 octets) le fait de vouloir supprimer l'instruction 21 fait "planter" le ZX 81! Sans vouloir supprimer cette instruction il est toujours possible de la modifier en utilisant les commandes LIST 21 puis EDIT. J'ai essayé sur plusieurs ZX 81 et le même phénomène s'est produit.

Le programme ci-après permet de générer une instruction REM avec le nombre de caractères exact que vous désirez. Il faudra au préalable initialiser les adresses 16507 et 16508 avec le nombre de caractères voulu. L'instruction 20 ne devra comporter aucun caractères.

```
20  REM (Newline).
```

	LD	BC, (16507)	ED	4B	7B	40
	LD	HL, 16554	21	AA	40	
ENCORE	PUSH	BC	C5			
	PUSH	HL	E5			
	LD	A, 61	3E	3D		
	CALL	ESPACE	CD	26	05	
	POP	HL	E1			
	POP	BC	C1			
	INC	HL	23			
	DEC	BC	0B			
	LD	A, C	79			
	OR	B	B0			
	JR	NZ, ENCORE	20	F1		
	LD	BC, (16507)	ED	4B	7B	40
	LD	HL, (16551)	2A	A7	40	
	ADD	HL, BC	09			
	LD	(16551), HL	22	A7	40	
	RET		C9			

Pour avoir une instruction REM de 1 K octets il suffira de rentrer les trois commandes suivantes :

— POKE 16507, 0 (1024 = 4 × 256 + 0)
— POKE 16508, 4
— RAND USR 16514

2.3. COMMENT SAUVEGARDER UN PROGRAMME EN LANGAGE MACHINE SITUÉ DANS LE HAUT DE LA MÉMOIRE

Pour pouvoir sauvegarder un programme écrit en langage machine situé dans le haut de la mémoire nous utiliserons un tableau de dimension suffisante pour contenir le programme. Avant d'effectuer la sauvegarde nous initialiserons ce tableau avec le contenu du haut de la mémoire. La première chose que nous réaliserons lors du rechargement sera la recopie du contenu de ce tableau dans le haut de la mémoire.

```
9000 LET R=PEEK 16388+256*PEEK 1
6389-1
9005 LET D=32767-R
9010 FAST
9015 DIM S(D)
9020 FOR I=1 TO D
9025 LET S(I)=PEEK (R+I)
9030 NEXT I
9035 PRINT "NOM?"
9040 INPUT W$
9045 CLS
9050 SAVE W$
9055 PRINT " SAVE (S) OU LOAD (L
) ?"
9060 INPUT Y$
9065 IF Y$="S" THEN STOP
9070 FOR I=1 TO D
9075 POKE (R+I),S(I)
9080 NEXT I
9085 PRINT "INITIALISATION RAMTO
P TERMINEE"
9090 SLOW
9095 STOP
```

Ce programme suppose que vous travaillez avec le bloc 16 K RAM.

La question "SAVE (S) ou LOAD (L)" permet d'éviter la recopie du tableau S dans le haut de la mémoire lors d'une sauvegarde.

Avant de recharger il faudra réinitialiser la variable système RAMTOP.

3

Programme d'aide à la mise au point de programmes écrits en langage machine

Un des inconvénients des programmes écrits en langage machine est que l'on est pratiquement désarmé pour leur mise au point. En effet, l'exécution d'un programme écrit en BASIC pourra toujours être arrêtée en appuyant sur la touche "BREAK" — si par exemple celui-ci boucle — alors que cela sera impossible pour un programme écrit en langage machine, la scrutation du clavier n'étant plus effectuée. Le seul recours dont on dispose alors pour arrêter le " frétillement " de l'image est de couper l'alimentation. Il ne vous reste plus alors qu'à vous replonger dans votre programme, sans aucune piste pour vous permettre de trouver l'erreur. Avouez que cela est plutôt frustrant !

Dans ce chapitre nous vous donnons un programme d'aide à la mise au point des programmes écrits en langage machine. Ce programme comporte des parties écrites en BASIC et en langage machine.

3.1. PROGRAMME D'AIDE A LA MISE AU POINT*

Ce programme comporte trois sous-programmes écrits en langage machine, dont le code hexadécimal est donné ci-après, implantés respectivement aux adresses 16514, 16565 et 16593. Pour mémoriser ces trois programmes, rentrez trois instructions REM comme suit :

3 REM XXX ← 45 fois → XXX

4 REM YY ← 22 fois → YY

5 REM ZZ ← 67 fois → ZZ

Il ne vous reste plus qu'à utiliser le programme de chargement suivant en changeant pour chaque sous programme les valeurs initiale et finale de N. Si vous avez fait une erreur il vous sera possible de revenir à l'octet précédent en appuyant sur la touche R ; l'adresse de cet octet sera alors affiché afin de vous indiquer clairement quel est l'octet attendu.

Remarque : Vous pouvez vérifier que les trois instructions REM ont le nombre de caractères voulu en lisant leur longueur. En effet les commandes.

PRINT PEEK 16511, PRINT PEEK 16562 et PRINT PEEK 16590

devront respectivement donner 47, 24 et 69 comme résultat.

```
9000 FOR N=16514 TO 16558
9010 INPUT A$
9020 SCROLL
9030 IF A$<>"R" THEN GOTO 9070
9040 LET N=N-1
9050 PRINT N;" ";
9060 INPUT A$
9070 PRINT A$
9080 POKE N,16*CODE A$+CODE A$(2
)-476
9090 NEXT N
9100 STOP
```

— *Programme de chargement* :

* Avec bloc 16K RAM.

```
?D
16514 16558

16514 11 03 00 0E 06 21 9D 40
16522 3A 3C 40 BE 28 08 19 0D
16530 20 F9 01 95 1F C9 23 4E
16538 23 46 C9 29 BD 1F 38 03
16546 20 32 44 20 2C 58 20 37
16554 6C 20 28 C6 20

?
```

```
?D
16565 16586

16565 ED 5B 40 40 2A 3E 40 ED
16573 4B 3C 40 C5 A7 ED 42 23
16581 E5 C1 E1 ED B0 C9

?
```

```
?D
16593 16659

16593 21 3E 40 7E 23 B6 2A 3C
16601 40 28 1B ED 73 51 40 E5
16609 01 03 00 2A 3E 40 E5 11
16617 1A 41 ED B0 E1 36 CD 23
16625 36 F8 23 36 40 E1 E9 31
16633 51 40 FD E5 DD E5 E5 D5
16641 C5 F5 01 03 00 21 1A 41
16649 ED 5B 3E 40 ED B0 ED 7B
16657 51 40 C9

?
```

Code hexadécimal des trois sous-programmes
écrits en langage machine.

```
   3 REM )▀ :▞5▌RNDUWRND▌C▒;$4 R
AND ▪= TAN 7??7TAN D▌35▀4M?4G?4R
?4FLEN 4
   4 REM GOSUB ?RNDRNDEYRND GOSU
B ?WRNDVAL ▌ GOSUB PI7 FAST AT L
PRINT GOSUB ▌TAN
   5 REM 5YRNDC. GOSUB ??RND FAS
T ▪▀ EYRND FAST ),INKEY$ GOSUB ▌
 LPRINT QLN 7Q SAVE 7QRND LPRINT
 DIM L?RND CLEAR FAST <> FAST FA
ST STR$ VAL PRINT ▪▀ 5,INKEY$ GO
SUB ?YRND GOSUB ▌ GOSUB ??RNDTAN

   6 REM XXX
   7 REM XXXXXXXXXXXXXXXXXXX0XXX
XXXXXX0XXXXXXXXX0XXXXXXXXX0XXXXX
XXXX0XXXXXXXXX0XXXXXXXXX0XXXXXXX
XX0XXXXXXXXX0XXXXXXXXXXXXXXX0XXX
XXXXXX0XXXXXXXXX0XXXXXXXXX0XXXXX
XXXX0XXXXXXXXX0XXXXXXXXX0XXXXXXX
XX0XXXXXXXXX0XXXXXXXXX0
8020 DIM R$(10,2)
8025 LET R$(1)=" F"
8030 LET R$(2)=" A"
8035 LET R$(3)=" C"
8040 LET R$(4)=" B"
8045 LET R$(5)=" E"
8050 LET R$(6)=" D"
8055 LET R$(7)="HL"
8060 LET R$(8)="IX"
8065 LET R$(9)="IY"
8075 LET W=16444
8080 DIM F(3)
8085 PRINT
8090 PRINT
8095 PRINT "?";
8100 INPUT C$
8102 SLOW
8105 CLS
8110 PRINT "?";C$(1)
8115 POKE 16444,CODE C$
8120 GOTO USR 16514
8125 LET N=2
8130 GOSUB 8395
8135 FOR I=F(1) TO F(2)
8140 PRINT F(1);" ";
8145 FOR J=0 TO 7
8150 LET E=PEEK F(1)
8155 GOSUB 8445
8160 PRINT " ";
8165 LET F(1)=F(1)+1
8170 IF F(1)>F(2) THEN GOTO 8085
8175 NEXT J
8180 PRINT
8185 NEXT I
8190 GOTO 8085
8195 LET N=1
```

```
8200 GOSUB 8395
8205 PRINT A;" ";
8210 LET E=PEEK A
8215 GOSUB 8445
8220 INPUT C$
8225 IF C$="." THEN GOTO 8085
8230 IF C$="" THEN GOTO 8245
8235 PRINT "-";C$( TO 2);
8240 POKE A,16*CODE C$+CODE C$(2
)-476
8245 PRINT " ";
8250 LET A=A+1
8255 GOTO 8210
8260 LET N=3
8265 GOSUB 8395
8270 RAND USR 16565
8275 GOTO 8085
8280 LET N=2
8285 GOSUB 8395
8290 RAND USR 16593
8295 GOTO 8085
8300 LET X=16453
8305 FOR I=1 TO 6
8310 PRINT R$(I);"=";
8315 LET E=PEEK X
8320 GOSUB 8445
8325 LET X=X+1
8330 PRINT " ";
8335 NEXT I
8340 FOR I=7 TO 9
8345 PRINT R$(I);"=";
8350 LET E=PEEK (X+1)
8355 GOSUB 8445
8360 LET E=PEEK X
8365 GOSUB 8445
8370 PRINT " ";
8375 LET X=X+2
8380 NEXT I
8385 GOTO 8085
8390 STOP
8395 FOR L=1 TO N
8400 INPUT A
8405 PRINT A;" ";
8410 POKE (W+2*(L-1)+1),INT (A/2
56)
8415 POKE (W+2*(L-1)),A-256*INT
(A/256)
8420 LET F(L)=A
8425 NEXT L
8430 PRINT
8435 PRINT
8440 RETURN
8445 PRINT CHR$ (INT (E/16)+28);
CHR$ (E-16*INT (E/16)+28);
8450 RETURN
```

1. Le code assembleur et l'explication des trois sous-programmes écrits en langage machine sont donnés dans l'annexe 1 de ce livre.

2. Ce programme utilise le buffer de l'imprimante pour stocker des paramètres. N'utilisez donc pas cette zone pour mémoriser vos propres paramètres.

3. L'instruction 7 REM x.....x peut contenir 206 octets, nous l'utiliserons pour essayer des petits programmes par la suite. L'adresse du premier octet disponible de cette instruction est 16675.

3.2. UTILISATION DU PROGRAMME

Après avoir rentré et sauvegardé sur cassette ce programme lancez son exécution par la commande RUN. Un point d'interrogation '?' est affiché en haut de l'écran pour vous inviter à rentrer une commande. Six commandes sont à votre disposition :

— D : "Dump" d'une zone mémoire.

— F : Sortie du programme.

— G : Exécution d'un programme écrit en langage machine.

— M : Recopie d'une zone mémoire.

— R : Affichage du contenu des registres.

— S : Modification d'une position mémoire.

Après avoir rentré une commande le programme vous demandera une, deux ou trois adresses (sauf pour les commandes F et R). Ces adresses sont des adresses décimales. Dès que la commande choisie a été exécutée vous avez la possibilité de spécifier une nouvelle commande.

3.2.1. "Dump" d'une zone mémoire — Commande D

```
D
ADR 1      ADR 2
```

La commande D affiche en hexadécimal le contenu de la zone mémoire débutant à l'adresse ADR 1 et finissant à l'adresse ADR 2. Les adresses ADR 1 et ADR 2 sont des adresses décimales.

Exemple: Dump de la zone mémoire commençant à l'adresse décimale 100 et finissant à l'adresse décimale 150.

```
?D
100 150

100 14 FF 08 3C FA 6D 00 28
108 02 08 C9 08 F5 C5 D5 E5
116 2A 0C 40 CB FC 76 D3 FD
124 DD E9 3F 3D 28 36 26 38
132 29 2B 2C 36 3C 2A 37 39
140 1D 1E 1F 20 21 1C 25 24
148 23 22 35

?
```

3.2.2. Sortie du programme d'aide de mise au point — Commande F

Cette commande ne nécessite aucune adresse et vous permet d'arrêter l'exécution du programme d'aide à la mise au point.

Son exécution provoque l'affichage du report 9/8390.

3.2.3. Exécution d'un programme écrit en langage machine — Commande G

G
ADR 1 ADR 2

La commande G permet de lancer l'exécution d'un programme écrit en langage machine en spécifiant un point d'arrêt — c'est-à-dire une adresse à laquelle l'exécution du programme sera arrêtée. Cette adresse ne sera pas obligatoirement le retour du sous-programme. Avec cette commande les registres primaires du microprocesseur Z 80 sont sauvegardés et pourront être consultés avec la commande R.

Si aucun point d'arrêt n'est désiré, il suffit de donner une adresse nulle à ADR 2. Dans ce cas il n'y aura pas de sauvegarde des registres.

Exemples:

```
? G
16675     16678
```

Lancer l'exécution du programme qui débute à l'adresse 16675 et arrêter cette exécution à l'adresse 16678.

```
? G
16675     0
```

Lancer l'exécution du programme qui débute à l'adresse 16675.

Remarque 1. Lorsque vous arrêtez l'exécution d'un programme par un point d'arrêt, vous ne pouvez pas relancer l'exécution à partir de ce point d'arrêt, il faut toujours relancer l'exécution à partir du début du programme.

Exemple: Vous avez écrit un programme en langage machine commençant à l'adresse 16675 et finissant à l'adresse 16690. Vous désirez mettre un point d'arrêt à l'adresse 16678, vous entrez donc la commande:

```
G
16675     16678
```

Supposons maintenant que vous désiriez mettre un point d'arrêt à l'adresse 16685, il faudra alors entrer la commande.

```
G                    et non    G
16675     16685                16678    16685
```

Remarque 2. Pour pouvoir sauvegarder le contenu des registres lorsque vous demandez un point d'arrêt, nous utilisons un " petit stratagème". Nous remplaçons *momentanément* les trois premiers octets situés à partir du point d'arrêt par un appel à un sous-programme particulier que nous appellerons SAUVEGARDE. Il faudra donc veiller à ne pas se brancher dans l'un de ces trois octets.

Exemple:

Dans un sous-programme écrit en langage machine, vous testez le contenu d'une variable. Si cette variable est nulle vous effectuez un retour au BASIC, si non vous continuez votre traitement. Votre programme en langage machine pourra être du type suivant:

Adresse décimale				
	DEBUT			
		JR	C, SUITE	; débranchement en 16701 si indicateur C à 1
		LD	A, (VARIABLE)	; test de la variable
		OR	A	; positionne indicateur (registre F)
16700		RET	Z	; Retour si zéro
16701	SUITE	INC	HL	; Suite du traitement
16702		LD	(HL), A	
16703		LD	B, NOMBRE	

Supposons maintenant que vous désiriez mettre un point d'arrêt juste avant le retour au BASIC pour connaître le contenu des registres. Vous mettez donc votre point d'arrêt à l'adresse 16700. Le programme d'aide à la mise au point remplacera le contenu des adresses 16700, 16701 et 16702 par l'instruction :

CALL SAUVEGARDE

Votre programme aura alors l'aspect suivant :

Adresse décimale				
	DEBUT			
		JR	C, SUITE	; débranchement en 16701 si flag C à 1
		LD	A, (VARIABLE)	
		OR	A	
16700		CALL	SAUVEGARDE	
16703		LD	B, NOMBRE	

mais l'instruction JR C, SUITE effectue toujours son débranchement à l'adresse 16701 qui contient maintenant les poids faibles de l'adresse du sous-programme SAUVEGARDE. Si le débranchement se réalise (et, avouons-le, ceci arrivera surement !) les poids faibles de cette adresse seront alors interprétés comme un code opération par le microprocesseur Z 80 ce qui pourra créer un mauvais fonctionnement du programme...

Il faudra donc faire attention en spécifiant un point d'arrêt.

3.2.4. Recopie d'une zone mémoire — Commande M

M
ADR 1 ADR 2 ADR 3

La commande M permet de recopier le contenu d'une zone mémoire commençant à l'adresse ADR 1 et finissant à l'adresse ADR 2, dans une nouvelle zone mémoire commençant à l'adresse ADR 3. La zone mémoire ADR 1 - ADR 2 n'est pas effacée. ADR 1, ADR 2 et ADR 3 sont des adresses décimales.

Exemple:

```
?D
16678 16686

16678 3D 3D 3D 3D 3D 3D 3D 1C
16686 3D

?
```

Dump de la zone mémoire avant exécution de la commande M.

```
?
?M
16514 16516 16680


?D
16678 16686

16678 3D 3D 11 03 00 3D 3D 1C
16686 3D

?
```

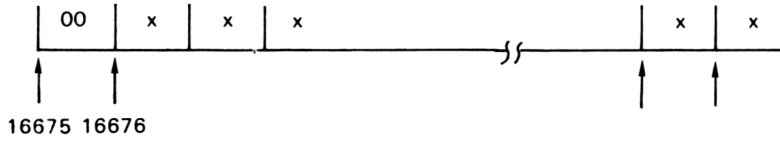
Commande M et dump de la zone mémoire après exécution de la commande M.

Remarque:

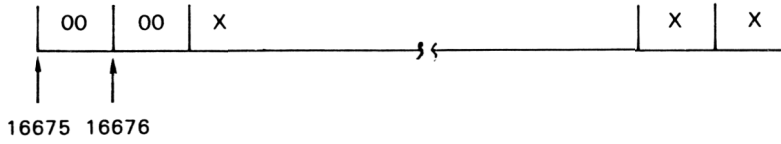
Il faudra veiller à ce que les zones émettrice et receptrice ne se chevauchent pas. En effet la commande:

```
M
16675     16685     16676
```

ne recopie pas la zone 16675 à 16685 à partir de l'adresse 16676 mais elle recopiera le contenu de l'adresse 16675 dans la zone 16676 à 16686! Prenons un exemple supposons que le contenu de l'adresse 16675 soit 00.



La recopie de la zone s'effectue octet par octet nous recopierons donc le premier octet de la zone émettrice dans le premier octet de la zone réceptrice. Le contenu de l'adresse 16676 deviendra ainsi 00.



Ensuite le deuxième octet de la zone émettrice (ie le contenu de l'adresse 16676 qui vient d'être mis à zéro) sera recopié dans le deuxième octet de la zone réceptrice (à l'adresse 16677 qui contiendra alors 00) et ainsi de suite. Finalement la zone mémoire 16675 à 16686 ne contiendra que des zéros.

Exemple:

```
?D
16675 16685

16675 00 3D 3D 3D 3D 3D 3D 3D
16683 3D 1C 3D

?
?M
16675 16685 16676


?D
16675 16686

16675 00 00 00 00 00 00 00 00
16683 00 00 00 00
```

3.2.5. Affichage du contenu des registres — Commande R

La commande R affiche le contenu hexadécimal des registres au dernier point d'arrêt. Cette commande ne nécessite aucune adresse.

Pour tester la commande R, utilisons le petit sous-programme suivant stocké à l'adresse 16675 (c'est-à-dire dans l'instruction 7 REM voir p. 50).

Pour charger ce programme utilisez la commande S du programme d'aide à la mise au point.

		Adresse	Code	hexa	
LD	BC, 0001	16675	01	01	00
INC	BC	78	03		
INC	BC	79	03		
INC	BC	80	03		
LD	DE, 0708 H	81	11	08	07
LD	HL, 0004	84	21	04	00
LD	A, 0	87	3E	00	
OR	A	89	B7		
RET		90	C9		

Ce programme n'a aucune fonction particulière. En spécifiant des points d'arrêts différents nous pourrons vérifier le contenu des registres. Ainsi en mettant un point d'arrêt à l'adresse 16679 nous devrons obtenir le contenu du registre C égal à 2 et celui du registre B égal à zéro.

```
?G
16675 16679


?R
 F=00  A=5F  C=02  B=00  E=45  D
=40 HL=411A IX=028F IY=4000

?
```

Un point d'arrêt à l'adresse 16687 nous donnera les valeurs suivantes :

C = 04, B = 00, E = 08, D = 07, HL = 0004

```
?G
16675 16687


?R
 F=28  A=67  C=04  B=00  E=08  D
=07 HL=0004 IX=0281 IY=4000

?
```

Un point d'arrêt à l'adresse 16690 nous donnera les mêmes valeurs pour les registres B, C, D, E, H et L et les valeurs suivantes pour l'accumulateur et les flags.

A = 00 F = 44

```
?
?G
16675 16690

?R
 F=44  A=00  C=04  B=00  E=08  D
=07 HL=0004 IX=028F IY=4000

?
```

L'instruction OR A remet à zéro les flags Report (C), Demi-report (H) et le flag interne N. D'autre part elle modifie selon le résultat de l'opération, les flags Zéro (Z), Signe (S), et Parité-débordement (P/V).

Le contenu de l'accumulateur étant nul lorsque l'instruction OR A a été exécutée nous aurons donc les valeurs suivantes :

flag S = 0
flag Z = 1

Vérifions cela. Le contenu du registre F est 44 en hexadécimal soit 01000100 en binaire. Nous en déduisons donc les valeurs suivantes pour les différents flags :

S = 0
Z = 1
H = 0
N = 0
C = 0

Remarque: Si vous n'êtes pas familier avec l'interprétation du registre Indicateur reportez-vous à "L'assembleur facile du Z 80" paru dans la même collection.

3.2.6. Examen et modification d'une position mémoire — Commande S

S
ADR 1

Après avoir rentré l'adresse décimale ADR 1, le contenu de cette position mémoire est affiché en hexadécimal. Il est alors possible de:

— passer à la position mémoire suivante en appuyant sur la touche " NEWLINE",

— modifier le contenu de cette position mémoire en entrant le nouveau contenu sous la forme de deux caractères hexadécimaux. Ainsi pour entrer la valeur 7 il faudra taper 07. Sur l'écran le nouveau contenu sera séparé de l'ancien par un tiret,

— d'arrêter la commande en appuyant sur les touches . (point) puis NEWLINE. Le point ne sera pas affiché.

Exemple: Modification du contenu de l'adresse décimale 16518. Le contenu initial 06 a été remplacé par 07.

```
?S
16514
16514 11 03 00 0E 06-07 21 9D
?S
16514
16514 11 03 00 0E 07 21 9D
?
```

Remarque: Cette commande pourra être utilisée pour rentrer un programme écrit en langage machine. En effet reprenons l'exemple du chapitre précédent: (tracé d'une diagonale).

Entrez la commande:

```
S
16675
```

le code hexadécimal 3D est affiché, vous pouvez alors entrer le premier code hexadécimal de l'instruction LD DE,0034 c'est-à-dire 11 et ainsi de suite. L'affichage de votre écran aura alors l'aspect suivant:

```
?S
16675

16675 3D-11 3D-22 3D-00 3D-2A 3D
-0C 3D-40 3D-23 3D-06 3D-14 3D-3
6 3D-86 3D-19 3D-10 3D-FB 3D-C9
3D

?
```

Entrez maintenant la commande:

```
G
16675      0
```

pour obtenir le tracé de la diagonale.

Remarque: Le programme d'aide à la mise au point occupe les adresses 16514 à 16659. Il se peut que vous désiriez utiliser ces adresses pour vos propres programmes. Il est possible, avec quelques modifications, de le transférer dans le haut de la mémoire. Je suppose que vous avez initialisé la variable système RAMTOP à 32000 avant d'avoir envoyer la commande NEW. Procédez alors de la manière suivante:

— Charger le programme et lancer son exécution.

— Recopier la zone 16514 à 16669 en 32000 (utiliser la commande M).

— Modifier les huit adresses suivantes (commande S), qui correspondent à des débranchements absolus.

32006	1B	au lieu de 9D
32007	7D	au lieu de 40
32103	98	au lieu de 1A
32104	7D	au lieu de 40
32112	76	au lieu de F8
32115	7D	au lieu de 40
32132	98	au lieu de 1A
32133	7D	au lieu de 41

— Entrer la commande F et modifier ainsi les lignes 8120, 8270, 8290 du programme Basic :

8120	GOTO	USR	32000
8270	RAND	USR	32051
8290	RAND	USR	32079

Vous pouvez maintenant supprimer les lignes 3, 4 et 5 du programme Basic.

Plus généralement si la variable système RAMTOP a été initialisée à x, les modifications sont :

adresse x + 6 mettre la valeur hexadécimale
(x + 27) – 256 * INT ((x + 27) / 256)

adresse x + 7 mettre la valeur hexadécimale INT ((x + 27)/256)

adresse x + 112 mettre la valeur hexadécimale
(x + 118) – 256 * INT ((x + 118)/256)

adresse x + 115 mettre la valeur hexadécimale INT ((x + 118)/256)

adresses x + 103 et x + 132 (x + 152) – 256 * INT ((x + 152)/256)

adresses x + 104 et x + 333 INT ((x + 152) / 256)

Programme BASIC :

8120	GOTO	USR	x
8270	GOTO	USR	(x + 51)
8290	GOTO	USR	(x + 79)

4

Comment scruter le clavier*

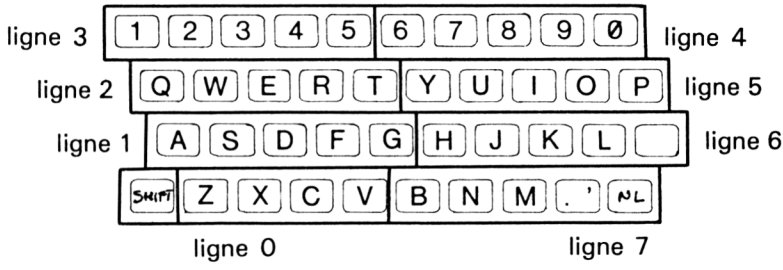
4.1. FONCTIONNEMENT DU SOUS-PROGRAMME DE LA ROM

Pour les besoins de l'interpréteur basic, la ROM de votre ZX 81 contient un sous-programme de scrutation du clavier dont le listing est le suivant :

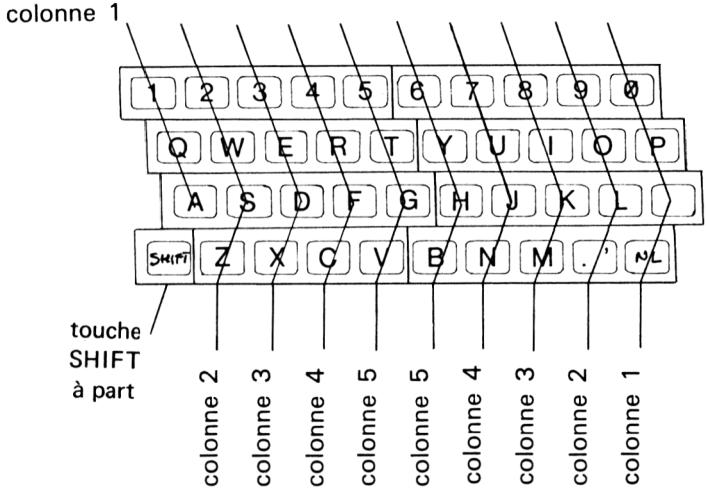
```
CLAVIER          LD    HL, FFFF H
                 LD    BC, FEFE H
                 IN    A, (C)
                 OR    01
LIGNE SUIVANTE   OR    E0 H
                 LD    D, A
                 CPL
                 CP    + 01
                 SBC   A, A
                 OR    B
                 AND   L
                 LD    L, A
                 LD    A, H
                 AND   D
                 LD    H, A
                 RLC   B
                 IN    A, C
                 JR    C, LIGNE SUIVANTE
                 RRA
                 RL    H

                 RET
```

* Reportez-vous aussi au livre "*Micro-ordinateurs, comment ça marche?*" paru dans la même collection.

Ce sous-programme débute à l'adresse hexadécimale 02BBH.



Les huit lignes du clavier du ZX81.



Les 5 colonnes du clavier.

Une des premières instructions de ce sous-programme est l'instruction IN A, (C). C'est cette instruction qui va nous permettre de lire l'état du clavier.

Le clavier du ZX 81 est divisé en huit lignes de cinq colonnes chacune (voir schéma ci-avant). L'instruction IN A, (C) suppose que l'adresse du périphérique sélectionné se trouve dans la paire de registres BC. Le registre C est initialisé à FE H et nous n'y toucherons pas. Par contre en agissant sur le contenu du registre B nous pourrons choisir une ligne particulière. En effet une ligne particulière du clavier sera sélectionnée en mettant à zéro le bit qui lui correspond dans le registre B. (Ceci est dû à la conception même du clavier du ZX 81 — l'explication en est donnée dans l'annexe 2 de ce livre.)

Ainsi pour sélectionner la ligne 0 le contenu du registre B sera:

1	1	1	1	1	1	1	0
bit 7							bit 0

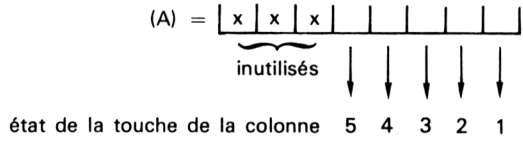
c'est-à-dire FE

pour sélectionner la ligne 1, ce sera:

1	1	1	1	1	1	0	1

c'est-à-dire FD, etc...

L'instruction IN A, (C) stockera le résultat lu, dans l'accumulateur. Celui-ci contiendra donc l'état des cinq touches situées sur la ligne sélectionnée par le registre B.



Une touche enfoncée apparaîtra comme un 0, une touche relachée comme un 1. Ainsi pour la ligne 1, l'état de l'accumulateur sera:

x	x	x	1	1	0	1	1
b7							b0

si la touche D est enfoncée.

x	x	x	0	1	1	1	1
b7							b0

si la touche G est enfoncée.

x	x	x	1	1	1	1	1
b7							b0

si aucune touche n'est enfoncée.

Revenons au sous-programme de scrutation du clavier. Ce sous-programme identifiera la touche appuyée dans les registres H et L de la façon suivante. H et L indiquent respectivement la colonne et la ligne auxquelles la touche appartient.

Si le contenu de L est	FE	: une touche de la ligne 0 a été enfoncée
	FD	: une touche de la ligne 1 a été enfoncée
	FB	: une touche de la ligne 2 a été enfoncée
	F7	: une touche de la ligne 3 a été enfoncée
	EF	: une touche de la ligne 4 a été enfoncée
	DF	: une touche de la ligne 5 a été enfoncée
	BF	: une touche de la ligne 6 a été enfoncée
	7F	: une touche de la ligne 7 a été enfoncée
	FF	: aucune touche enfoncée.

Si le contenu de H est	FD	: une touche de la colonne 1 a été enfoncée
est	FD	: une touche de la colonne 2 a été enfoncée
est	F7	: une touche de la colonne 3 a été enfoncée
est	EF	: une touche de la colonne 4 a été enfoncée
est	DF	: une touche de la colonne 5 a été enfoncée
est	FF	: aucune touche enfoncée.

Ceci en admettant que nous n'appuyons que sur une seule touche (nous ignorons donc — momentanément — la touche SHIFT qui est particulière).

Supposons que nous appuyons sur la touche D. Le sous-programme commence la scrutation du clavier par la ligne 0 (registre B initialisé à FE H); après avoir lu l'état de cette ligne, le ZX 81 ignorera l'état de la touche SHIFT — pour cela son état est forcé à 1 (OR 01) comme si cette touche était relachée — puis le mot d'état de la ligne ne comportant que cinq bits valides, les trois bits restants sont forcés à 1 (OR E0 H). Le contenu de l'accumulateur est donc FF (car sur la ligne 0 aucune touche n'est enfoncée).

Examinons maintenant le rôle de la séquence:

```
CPL
CP      01
SBC     A, A
```

Cette séquence initialisera l'accumulateur à 00 si une touche est enfoncée et à FF dans le cas contraire. En effet regardons les deux cas parallèlement:

Scrutation ligne 0 pas de touche enfoncée		Scrutation ligne 1 touche D enfoncée
(A) = FF H	avant CPL	(A) = FB H
(A) = 00	après CPL	(A) = 04
(A) = 00 flag C = 1 car (A) ⩽ 01	après comparaison	A = 04 flag C = 0 car (A) ⩾ 01
(A) = FF	après soustraction avec flag C	(A) = 0

Ainsi l'instruction suivante OR B n'aura aucun rôle dans le cas où il n'y a pas eu d'appui touche pour la ligne. Par contre si un appui touche a été détecté le contenu du registre B sera recopié dans l'accumulateur. Or le registre B est celui qui permet de sélectionner les différentes lignes du clavier. Ce résultat est ensuite rangé dans le registre L. (L'instruction AND L sert uniquement à tenir compte des appuis touches qui auraient pu être effectués auparavant — cas où vous appuyez simultanément sur deux touches situées sur des lignes différentes.

Le contenu du registre H est déterminé beaucoup plus simplement : c'est la recopie de l'état des touches de la ligne considérée en utilisant l'instruction AND ; ceci de façon à tenir compte des appuis touches simultanés. A la fin du sous-programme le contenu du registre H sera décalé de un bit à gauche (RL H).

Intéressons-nous maintenant à la touche SHIFT ! En réalité l'état des touches de la ligne 0 est lu deux fois. Une première fois dès le début du sous-programme et une seconde fois à la fin. Cependant cette seconde fois, le débranchement à LIGNE SUIVANTE ne sera pas effectué. Les deux instructions :

```
RRA
RL      H      sont donc exécutées
```

Regardons l'effet de ces instructions dans le cas où nous avons appuyé sur la touche SHIFT :

avant le RRA les contenus de l'accumulateur, du flag C et du registre H sont :

(A) = x x x 1 1 1 1 0 indéterminés — touche SHIFT appuyée	C = 0	H colonne de la touche appuyée

après le RRA ils sont :

(A) = 0 x x x 1 1 1 1	C = 0 touche SHIFT appuyée	H colonne de la touche appuyée

et après le RL H ils sont :

(A) = 0 x x x 1 1 1 1 C = ? H = x x x x x x x 0

Ainsi le fait d'appuyer sur la touche SHIFT force le bit 0 du registre H à zéro. Nous pouvons donc maintenant dresser la table des valeurs du registre H lorsque il y a eu un appui sur la touche SHIFT.

Si le contenu de H est	FC	: SHIFT + une touche de la colonne 1 ont été enfoncées
	FA	: SHIFT + une touche de la colonne 2 ont été enfoncées
	FG	: SHIFT + une touche de la colonne 3 ont été enfoncées
	E7	: SHIFT + une touche de la colonne 4 ont été enfoncées
	D7	: SHIFT + une touche de la colonne 5 ont été enfoncées
	FE	: la touche SHIFT seule a été enfoncée

Vérifions ceci avec le programme suivant. Ce programme sera rentré à l'adresse 16675 et chargé avec le programme d'aide à la mise au point en utilisant la commande S.

DEBUT	CALL	CLAVIER	CD	BB	02
	LD	A, H	7C		
	AND	L	A5		
	INC	A	3C		
	JR	Z, DEBUT	28	F8	
	RET		C9		

Lancer l'exécution du programme par la commande G en mettant un point d'arrêt à l'adresse 16683 puis appuyer sur une des touches du clavier. Ensuite examiner les registres H et L avec la commande R.

Exemple: Résultat obtenu en appuyant sur la touche J (ligne 6 colonne 4) nous vérifions bien ainsi que:

H = EF et L = BF

```
?R
 F=B0  A=B0  C=FE  B=FE  E=45  D
=FF HL=EFBF IX=028F IY=4000

?
```

Recommencer en changeant de touche.

Remarque: la séquence:

```
LD    | A, H
AND   | L
INC   | A
JR    | Z, DEBUT
```

sert à tester si il y a eu un appui touche. En effet, lorsqu'aucune touche n'a été enfoncée les registres H et L contiennent la valeur FF H. Le AND entre ces deux valeurs redonnera donc FF H (c'est-à-dire – 1) en incrémentant cette valeur de un, nous obtiendrons zéro comme résultat. Si par contre un appui touche est détecté, l'un au moins des deux registres H et L aura une valeur différente de FF H et l'incrémentation fournira alors un résultat différent de zéro.

4.2. COMMENT RETROUVER LE CODE DU CARACTÈRE CORRESPONDANT A LA TOUCHE APPUYÉE

Nous avons maintenant un code dans les registres H et L identifiant d'une façon unique la touche qui a été appuyée. Mais reconnaissons-le, ce code n'est pas très explicite et il serait beaucoup plus pratique de pouvoir obtenir dans un registre la valeur 26H si par exemple nous avons appuyé sur la touche A.

Un sous-programme de la ROM existe et réalise ce travail. Appelons le CODE. Ce sous-programme se trouve à l'adresse 07BD H. Pour utiliser correctement ce sous-programme, il faudra charger dans les registres B et C, le code obtenu dans les registres H et L après une scrutation du clavier. A la fin de l'exécution de ce sous-programme la paire de registres HL contiendra l'adresse où se trouve le code du caractère de la touche appuyée.

Exemple: Reprenons l'exemple précédent mais en le complétant de manière à obtenir le code du caractère correspondant à la touche appuyée dans le registre A.

DEBUT	CALL	CLAVIER	CD	BB	02
	LD	A, H	7C		
	AND	L	A5		
	INC	A	3C		
	JR	Z, DEBUT	28	F8	
	PUSH	HL	E5		
	POP	BC	C1		
	CALL	CODE	CD	BD	07
	LD	A, (HL)	7E		
	RET		C9		

Lancer le programme par G en mettant un point d'arrêt à l'adresse 16689 (l'adresse du début est 16675). Puis appuyez sur la touche A. En venant examiner les registres vous devez obtenir A = 26H qui est bien le code du caractère A (voir manuel SINCLAIR p. 182. Code A = 38 = 26 en hexadécimal).

```
?R
 F=41  A=26  C=FE  B=FE  E=05  D
=00 HL=0082 IX=028F IY=4000

?
```

Recommencer avec d'autres touches. Toutefois n'utilisez pas la touche SHIFT, vous auriez quelques problèmes !...

Le programme suivant va vous permettre de déplacer un curseur de droite à gauche sur la première ligne de l'écran. Pour aller à droite appuyer sur la touche 8 (sans SHIFT), à gauche sur la touche 5 et sur une autre touche pour arrêter le programme (sauf SHIFT qui n'est pas prise en considération).

Pour rentrer ce programme utilisez le programme d'aide à la mise au point avec la commande S à l'adresse 16675. Lancer son exécution avec la commande G sans point d'arrêt (G, 16675, 0). Assurez-vous cependant que vous travaillez en mode SLOW (si vous lancez l'exécution en mode FAST l'écran se brouillera complètement et ne reprendra son aspect normal qu'à la fin de l'exécution du programme).

	LD	D, 31	16	1F	
	LD	HL, (16396)	2A	OC	40
	INC	HL	23		
	LD	(HL), 176 *(K en inversion vidéo)*	36	BO	
	LD	(16398), HL	22	OE	40
LECTURE	PUSH	DE	D5		
	CALL	CLAVIER	CD	BB	02
	LD	A, L	7D		
	INC	A	3C		
	POP	DE	D1		
	JR	Z, LECTURE	28	F7	
	PUSH	DE	D5		
	PUSH	HL	E5		
	POP	BC	C1		
	CALL	CODE	CD	BD	07
	POP	DE	D1		
	LD	A, (HL)	7E		
	LD	HL, (16398)	2A	OE	40
	CP	33	FE	21	
	JR	Z, GAUCHE	28	OF	
	CP	36	FE	24	
	JR	Z, DROITE	28	01	
	RET		C9		
DROITE	INC	D	14		
	DEC	D	15		
	JR	Z, LECTURE	28	DF	
	LD	(HL), 128 *(espace en inversion vidéo)*	36	80	
	INC	HL	23		
	DEC	D	15		
	JR	FIN	18	09	
GAUCHE	LD	A, D	7A		
	CP	31	FE	1F	
	JR	Z, LECTURE	28	D4	
	INC	D	14		
	LD	(HL), 00	36	00	
	DEC	HL	2B		
FIN	LD	(HL), 176 *(K en inversion vidéo)*	36	BO	
	LD	(16398), HL	22	OE	40
	JR	LECT	18	C9	

Après l'appel du sous-programme de scrutation du clavier, un test est réalisé de façon à savoir si un appui touche a eu lieu ou non. En effet le programme de recherche du code du caractère (CODE) suppose que le code contenu dans la paire de registres BC est un code "valide". D'autre part ce programme détruit les registres D et E. Il faudra donc les sauvegarder.

Dans ce petit programme le registre D est utilisé pour mémoriser la position du curseur sur la ligne. Pour chaque déplacement à droite son contenu sera décrémenté de un et inversement pour chaque déplacement à gauche son contenu sera incrémenté de un. Ce registre est initialisé à 31 au début du programme. En effet une ligne de l'écran ne peut contenir que 32 caractères nous ne pourrons donc effectuer que 31 déplacements sur la droite (le curseur occupant la place d'un caractère). Ainsi quand le contenu du registre D sera nul il ne faudra plus faire progresser le curseur. Si nous le faisions, nous écraserions le caractère NEWLINE qui indique la fin d'une ligne d'affichage ce qui nous obligerait à couper l'alimentation pour réobtenir un ZX absolument docile répondant immédiatement au moindre appui touche... C'est la raison du test.

```
DROITE | INC  |  D
       | DEC  |  D
       | JR   |  Z, LECTURE
```

De même lorsque l'on se déplace vers la gauche, il faudra s'arrêter lorsque le contenu du registre D sera égal à 31 car cela signifiera que nous sommes revenus au début de la ligne. Vous pouvez remarquer aussi la méthode employée pour tester s'il y a eu un appui touche.

4.3. AUTRE MÉTHODE DE SCRUTATION DU CLAVIER

Suivant les programmes que l'on réalise il n'est pas toujours utile de scruter toutes les touches du clavier. Par exemple, dans certains jeux impliquant le déplacement horizontal d'un curseur, seul l'état des touches 5 et 8 est intéressant. Il existe une méthode plus directe permettant de venir scruter uniquement ces deux touches. En effet, nous avons vu précédemment que pour scruter une ligne du clavier il fallait mettre le

bit qui lui correspond à zéro dans le registre B. La touche 5 se trouve sur la ligne 4 nous initialiserons donc le registre B à F7 H. Si au moment de la lecture un appui sur la touche 5 a lieu le contenu de l'accumulateur sera :

(A) = x x x 0 1 1 1 1

Masquons ce résultat avec :

0 0 0 1 0 0 0 0 (= 10 H)

c'est-à-dire un octet n'ayant qu'un seul bit à 1, ce bit étant celui de la colonne à laquelle appartient la touche à scruter. Il ne restera plus qu'à tester le flag Zéro (Z).

Z = 0 la touche était enfoncée.
Z = 1 la touche n'était pas enfoncée.

Pour scruter la touche 8, nous procéderons de même en prenant la valeur 04 pour le masque.

Ainsi si nous reprenons notre programme de déplacement du curseur nous pouvons remplacer la séquence d'instructions entre les deux étiquettes LECTURE et DROITE (première instruction PUSH DE, dernière instruction RET) par la séquence suivante.

LECTURE	LD	BC, F7FE H	01	FE	F7
	IN	A, (C)	ED	78	
	AND	10 H	E6	10	
	JR	Z, GAUCHE	28	1E	
	LD	B, EFH	06	EF	
	IN	A, (C)	ED	78	
	LD	E, A	5F		
	AND	04 H	E6	04	
	JR	Z, DROITE	28	0B	
	LD	A, E	7B		
	AND	01H	E6	01	
	JR	NZ, LECTURE	20	E9	
	RET		C9		

Nous serons par contre obligés de nous fixer une touche pour demander l'arrêt de l'exécution du sous-programme en langage machine. Ce sera la touche 0 (zéro).

Modifions le programme précédent avec la commande S. Cette version est un peu moins longue.

Avec cette méthode nous n'utilisons pas le sous-programme CODE qui peut être cause de problèmes si le code contenu dans la paire de registres BC n'est pas "valide" c'est-à-dire si il n'y a pas eu d'appui touche et donc (BC) = FFFFH — ou si il n'y a eu qu'un appui touche sur la touche SHIFT. — et donc (BC) = FEFF H. Regardons comment fonctionne le sous-programme CODE pour tenter d'expliquer cela. Son listing est le suivant:

```
CODE    | LD    | D, 00
        | SRA   | B
        | SBC   | A, A
        | OR    | 26H
        | LD    | L, 05
        | SUB   | L
CODE 2  | ADD   | A, L
        | SCF   |
        | RR    | C
        | JR    | C, CODE 2
        | INC   | C
        | RET   | NZ
        | LD    | C,B
        | DEC   | L
        | LD    | L, 01
        | JR    | NZ, CODE2
        | LD    | HL, 0007D H
        | LD    | E, A
        | ADD   | HL, DE
        | RET   |
```

Pour retrouver le code du caractère le ZX 81 s'adresse dans une table. Cette table est divisée en plusieurs parties:

1 — les codes des caractères sans SHIFT (A, B, C, ...).
2 — les codes des caractères avec SHIFT (*, →, ←, ...).
3 — les codes des fonctions (USR, INT, ABS, ...),
4 — les codes des caractères graphiques,
5 — les codes des instructions (GOTO, GOSUB, etc...).

Seules les deux premières parties de cette table nous intéressent. Chacune de ces deux parties est organisée de la même manière. Nous trouverons d'abord les codes des quatre touches de la ligne 0 (la touche SHIFT étant particulière) puis les codes des cinq touches de la ligne 1 (A, S, D, F, G) puis ceux de la ligne 2, etc...

Rentrons maintenant dans le détail du programme.

```
SRA  | B
SBC  | A, A
OR   | 26H
```

Ces trois instructions déterminent si la touche SHIFT a été appuyée. En effet si il n'y a pas eu d'appui sur la touche SHIFT le bit de poids faible du registre B est à 1, le décalage positionnera donc le report C à 1 — le résultat de la soustraction donnera donc FF H (– 1) rendant l'instruction OR 26H inopérante. Par contes si SHIFT a été appuyé le bit de poids faible du registre B est zéro, la soustraction aura alors un résultat nul (A) = 00. L'instruction OR 26H forcera donc l'accumulateur à 26H. Ce qui nous adressera dans la partie de la table des codes des caractères avec SHIFT.

La séquence d'instructions

```
CODE 2 | ADD  |  A, L
       | SCF  |
       | RR   |  C
       | JR   |  C, CODE 2
```

recherche sur quelle ligne l'appui touche a eu lieu et incrémente l'accumulateur de cinq à chaque passage dans la boucle. Si il y a eu un appui touche, l'un des bits du registre C sera à zéro donc en faisant des décalages à droite ce bit arrivera dans le report provoquant ainsi la sortie de la boucle. Mais s'il n'y a pas eu d'appui touche ou s'il n'y a eu qu'un appui sur la touche SHIFT le registre C sera initialisé à FF H (donc aucun bit à zéro). Nous bouclerons donc... jusqu'au moment où nous débrancherons! Cette boucle sera réutilisée pour détecter la touche appuyée, l'accumulateur étant incrémenté de 1 à chaque passage.

Remarquez comment sont différenciées la recherche de la ligne et la recherche de la touche à la sortie de la boucle.

```
LD   | C, B
DEC  | L
LD   | L, 01
JR   | NZ, CODE 2
```

Si l'on vient d'effectuer la recherche de la ligne le contenu du registre L est 05 donc l'instruction DEC L mettra le flag Zéro à zéro et nous repartirons dans la boucle pour la recherche de la touche. Par contre si nous venons d'effectuer la recherche de la touche le contenu du registre L est 01. L'instruction DEC L positionne donc le flag Zéro à 1 et le programme continuera en séquence.

Les deux instructions:

```
INC  | C
RET  | NZ
```

servent à détecter si vous avez appuyé sur plusieurs touches simultanément.

5

Maitriser le buffer d'affichage

Bien maîtriser l'affichage est à la base de tout jeu demandant un graphique un peu " évolué ". Nous procéderons en deux parties. D'abord nous verrons comment imprimer une suite de caractères en utilisant les sous-programmes de la ROM et ensuite comment travailler directement dans le buffer d'affichage.

5.1. COMMENT AFFICHER DES CARACTÈRES

Un sous-programme de la ROM permet d'afficher un caractère. Ce sous-programme se situe à l'adresse 0808 H. Appelons-le AFFICHAGE. Pour l'utiliser le code du caractère à afficher devra se trouver dans le registre A. Ce sous-programme ne vérifie pas si le code contenu dans l'accumulateur correspond à un caractère valide, il faudra donc s'en assurer. De plus, il modifie les contenus des paires de registres BC, DE et HL.

Exemple: Le programme ci-dessous affichera le caractère A.

LD	A, 26H	3E	26	
CALL	AFFICHAGE	CD	08	08
RET		C9		

Pour vérifier si l'accumulateur contient un caractère affichable nous pouvons utiliser la méthode suivante :

PUSH	AF	FS		
RES	7, A	CB	BF	
CP	40 H	FE	40	
RET	NC	D0		
POP	AF	F1		
CALL	AFFICHAGE	CD	08	08
RET		C9		

En effet les caractères "affichables" ont un code compris entre 0 et 63 et entre 128 et 191. Cette deuxième série est la première série en inversion vidéo. L'instruction RES 7, A remettra le bit 7 de l'accumulateur à zéro, ce qui revient, en réalité, à lui soustraire 128. Ainsi si le code correspond à un caractère valide il sera alors inférieur à 64. Le test CP 40 H mettra le flag C à zéro si ce code est supérieur à 64. Il ne nous restera plus qu'à tester le flag.

On peut aussi utiliser l'instruction RST 10 H en mettant le code du caractère à afficher dans l'accumulateur. En effet, vous savez que l'instruction RST vous branche à un endroit précis de la mémoire. Dans la ROM du ZX 81, RST 10 H est affecté à l'affichage des caractères. Nous remplacerons donc l'instruction CALL AFFICHAGE par RST 10 H.

Exemple : Le programme suivant remplira les premières lignes de l'écran avec le caractère X :

	LD	B, O	06	00
	LD	A, 61	3E	3D
CONT	RST	10 H	D7	
	DJNZ	CONT	10	FD
	RET		C9	

Avec cette deuxième méthode les registres sont préservés. Ces deux sous-programmes utilisent la variable système DF-CC* pour connaître l'endroit de l'écran où le caractère doit être affiché. Cette variable système est incrémentée chaque fois qu'un caractère est affiché. Ainsi si nous désirons effacer un caractère il faudra modifier auparavant le contenu de cette variable.

* Voir manuel SINCLAIR p. 178.

Exemple: Le programme suivant affiche 16 caractères. Appuyez sur une touche, le cinquième caractère sera effacé et le caractère Y imprimé après les 16 précédents. Rentrer ce programme à l'adresse 16675.

	LD	B, 16	06	10	
	LD	A, 61	3E	3D	
CONT	RST	10 H	D7		
	DJNZ	CONT	10	FD	
LECTURE	CALL	CLAVIER	CD	BB	02
	LD	A, L	7D		
	INC	A	3C		
	JR	Z, LECTURE	28	F9	
	LD	HL, (DF-CC)	2A	0E	40
	LD	DE, – 12	11	F4	FF
	PUSH	HL	E5		
	ADD	HL, DE	19		
	LD	(DF-CC), HL	22	0E	40
	LD	A, 00	3E	00	
	RST	10 H	D7		
	POP	HL	E1		
	LD	(DF-CC), HL	22	0E	40
	LD	A, 62	3E	3E	
	RST	10 H	D7		
	RET		C9		

Lancer l'exécution de ce programme en mode SLOW. Ce moyen ne pourra être utilisé que dans des cas particuliers. Si l'on désire pouvoir afficher un caractère n'importe où sur l'écran, il sera plus commode de spécifier les numéros de la colonne et de la ligne dans les registres B et C et d'utiliser le sous-programme débutant à l'adresse 08F5H dans la ROM. Ce sous-programme se chargera de calculer l'adresse dans le buffer d'affichage du caractère à éditer.

Exemple: Avec le programme suivant nous afficherons le caractère A à la 30e colonne de la 10e ligne de l'écran.

LD	B, 10	06	0A	
LD	C, 30	0E	1E	
CALL	08F5 H	CD	F5	08
LD	A, 38	3E	26	
RST	10 H	D7		
RET		C9		

Remarque: Les deux instructions LD B,10 et LD C, 30 auraient pu être condensées en:

LD	BC, 2590	01	1E	0A

Jusqu'à présent nous n'avons pas encore fait de sous-programme pour afficher une chaîne de caractères. Un tel sous-programme existe dans la ROM et se trouve à l'adresse 0B6B H. Les paires de registres BC et DE devront être respectivement initialisées avec la longueur et l'adresse de la chaîne de caractères.

Exemple: Faisons imprimer le texte: "ESSAI". Le programme sera chargé en 16675, l'adresse du premier caractère du mot ESSAI sera donc 16685.

	LD	BC, 0005	01	05	00		
	LD	DE, 16685	11	2D	41		
	CALL	OB6B H	CD	6B	0B		
	RET		C9				
DEFM	ESSAI		2A	38	38	26	2E

Pour notre exemple nous avons choisi un message très court. Le programme suivant va vous permettre d'éditer n'importe quelle chaîne de caractères. Il vous suffira simplement de l'initialiser avec l'instruction Basic LET. Avant de lancer l'exécution du programme en langage machine, il faudra mettre le nom de la variable alphanumérique dans la variable système située à l'adresse 16417.

	LD	BC, 0000	01	00	00
	LD	A, (16417)	3A	21	40
	AND	1F H	E6	1F	
	LD	D, A	57		
	LD	HL, (16400)	2A	10	40
RECHERCHE	ADD	HL, BC	09		
	LD	A, (HL)	7E		
	BIT	7, A	CB	7F	
	JR	Z, TEST 4	28	23	
TEST 2	BIT	6, A	CB	77	
	JR	NZ, TEST 3	20	0F	
	BIT	5, A	CB	6F	
	JR	Z, TABLEAU	28	14	

variable numérique dont le nom comporte plusieurs caractères

VARIABLE	INC	HL	23		
	LD	A, (HL)	7E		
	BIT	7, A	CB	7F	
	JR	Z, VARIABLE	28	FA	
	LD	BC, 0006	01	06	00
	JR	RECHERCHE	18	E7	
TEST	BIT	5, A	CB	6F	
	JR	Z, TABLEAU	28	05	
; variable de boucle					
	LD	BC, 0018	01	12	00
	JR	RECHERCHE	18	DE	
; tableau numérique ou alphanumérique					
TABLEAU	INC	HL	23		
	LD	C, (HL)	4E		
	INC	HL	23		
	LD	B, (HL)	46		
	INC	HL	23		
	JR	RECHERCHE	18	D7	
TEST 4					
	BIT	5,A	CB	6F	
	JR	Z, CHAINE	28	05	
	LD	BC, 0006	01	06	00
	JR	RECHERCHE	18	CE	
; vérifie si il s'agit de la bonne chaîne de caractère					
CHAINE	AND	AF H	E6	1F	
	CP	D	BA		
	INC	HL	23		
	LD	C, (HL)	4E		
	INC	HL	23		
	LD	B, (HL)	46		
	INC	HL	23		
	JR	NZ, RECHERCHE	20	C4	
; la bonne chaîne de caractères a été trouvée: édition					
	EX	DE, HL	E3		
	CALL	0B6B H	CD	6B	0B
	RET		C9		

Comment fonctionne ce programme.Vous vous rappelez que chaque variable est identifiée par son nom et par un code sur 3 bits nous donnant son type (variable de boucle, chaîne alphanumérique, tableau, etc...). Nous testons donc ce code. Si le premier bit (ie le bit de poids forts) est à 1, cela signifiera que la variable que nous sommes en train de traiter est soit une variable numérique dont le nom ne comporte qu'une seule lettre, soit une chaîne de caractères. (Reportez-vous au manuel SINCLAIR pp. 172 à 174). Il nous suffira ensuite de tester le bit 5 afin de connaître le type exact de la variable. Si le bit 5 est à zéro alors la variable est une chaîne de caractères. Nous comparerons alors son nom au nom de la variable demandée par l'utilisateur. Si il y a concordance nous l'imprimerons.

De même si le premier bit est à un nous serons en train de traiter :

— soit une variable numérique dont le nom comporte plusieurs caractères,

— soit un tableau (numérique ou alphanumérique),

— soit une variable de contrôle de boucle.

Nous testerons ensuite les bits 6 et 5 pour différencier chacun des cas.

Même si ces variables ne sont pas du type recherché, il faut absolument les tester pour connaître leur longueur de manière à pouvoir pointer directement sur la variable suivante dans la zone des variables.

Pour essayer ce programme, nous utiliserons le programme d'aide à la mise au point. Après l'avoir chargé rajoutez lui les instructions :

```
8437  LET Z$ = "ESSAI DU PROGRAMME"
8015  POKE 16417, 63
```

L'instruction POKE range le code de la lettre Z dans la variable système 16417. Lancez l'exécution du programme d'aide à la mise au point et chargez le programme en langage machine à partir de l'adresse 16675.

Remarque : Nous aurions pu utiliser les instructions de décalage à gauche à la place de l'instruction BIT. Il aurait alors fallu tester le flag C au lieu du flag Z.

5.2. TRAVAILLER DIRECTEMENT DANS LE BUFFER D'AFFICHAGE

Au lieu d'utiliser les sous-programmes de la ROM nous pouvons venir "écrire" directement dans le buffer d'affichage. Cependant avant d'utiliser cette méthode revoyons la structure du buffer d'affichage.

5.2.1. Structure du buffer d'affichage

La structure du buffer d'affichage est liée à la présence du bloc d'extension mémoire 16 K RAM.

— *Avec le bloc extension mémoire*: Chaque ligne de l'écran occupe dans le buffer d'affichage 33 caractères (32 caractères affichables et un caractère NEWLINE). Cette place sera reservée en mémoire quelque soit l'affichage même s'il n'y a rien à afficher.

— *Sans le bloc extension mémoire*: A l'initialisation chaque ligne de l'écran sera réduite dans le buffer d'affichage. *au seul caractère NEWLINE.* Un écran vide est alors constitué de 25 caractères NEWLINE. Dans ce cas, le ZX 81 cherche à économiser le maximum de mémoire en ne prenant pas en compte les caractères espaces inutiles. Le programme BASIC suivant permet de vérifier cela:

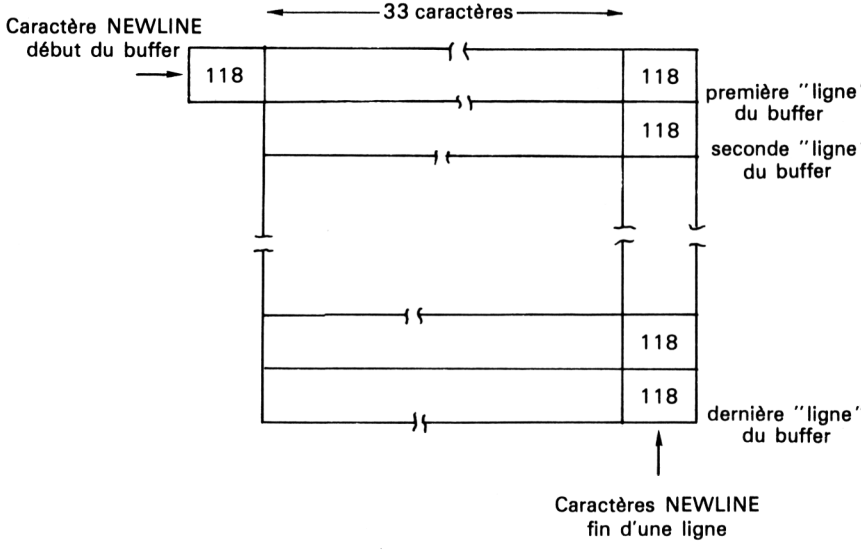
```
   4 PRINT "AA"
   6 PRINT "BBB"
  10 LET A=PEEK 16396+256*PEEK 1
6397
  20 DIM C(30)
  30 FOR I=1 TO 30
  40 LET C(I)=PEEK (I+A-1)
  50 NEXT I
  60 FOR I=1 TO 30
  70 PRINT C(I);" "
  80 NEXT I
  90 STOP

AA
BBB
118 38 38 118 39 39 39 118 118 1
18 118 118 118 118 118 118 118 1
18 118 118 118 118 118 118 118 1
18 118 118 118 118
```

Nous pouvons remarquer que le premier caractère du buffer d'affichage est le caractère NEWLINE. Sur la première ligne, comme il n'y a plus rien à afficher après les deux caractères A, les caractères espaces sont supprimés ; nous trouvons donc le caractère NEWLINE juste après le deuxième 'A'.

Ainsi, il faudra lorsque l'on travaille sans bloc extension mémoire, "étendre" le buffer d'affichage. Vous risqueriez sinon d'écraser un des caractères NEWLINE. Vous pouvez facilement imaginer la suite !...

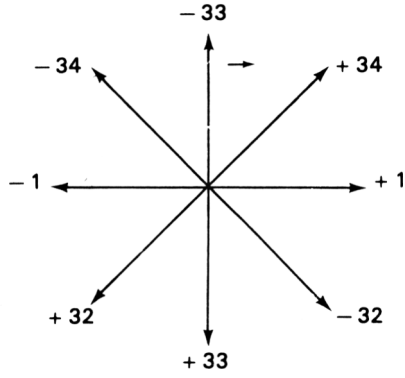
5.2.2. Applications

L'écran qui est connecté à votre ZX 81 est en réalité le reflet du contenu du buffer d'affichage. Ainsi faire se déplacer un caractère sur l'écran revient à le stocker en différents endroits du buffer d'affichage en l'effaçant de l'endroit où il se trouvait précédemment. Pour mieux "voir" les déplacements que nous allons réaliser représentons-nous le buffer d'affichage comme une superposition de 24 lignes de 33 caractères (le 33e caractère étant le caractère NEWLINE — voir schéma ci-dessous), et non comme une suite quelconque d'octets.



Nous nous apercevons ainsi que pour faire passer un caractère sur la ligne inférieure, il faudra venir le stocker 33 octets plus loin que son adresse initiale dans le buffer d'affichage.

Nous pouvons représenter les déplacements sur l'écran par le schéma suivant. Les chiffres indiqués correspondent au déplacement à effectuer dans le buffer d'affichage.



Exemple :

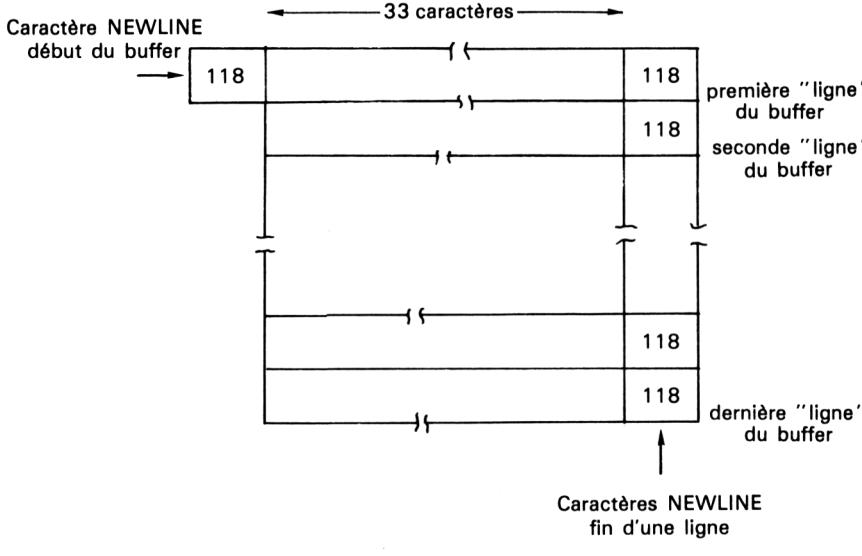
Le programme suivant affiche un caractère agrandi sur huit lignes et huit colonnes. Chargez-le à l'adresse 16675, mettez-vous en mode SLOW, lancez son exécution et appuyez sur une touche de votre choix. Si celle-ci ne correspond pas à un caractère affichable, elle sera ignorée par le programme.

Nous pouvons remarquer que le premier caractère du buffer d'affichage est le caractère NEWLINE. Sur la première ligne, comme il n'y a plus rien à afficher après les deux caractères A, les caractères espaces sont supprimés ; nous trouvons donc le caractère NEWLINE juste après le deuxième 'A'.

Ainsi, il faudra lorsque l'on travaille sans bloc extension mémoire, "étendre" le buffer d'affichage. Vous risqueriez sinon d'écraser un des caractères NEWLINE. Vous pouvez facilement imaginer la suite !...

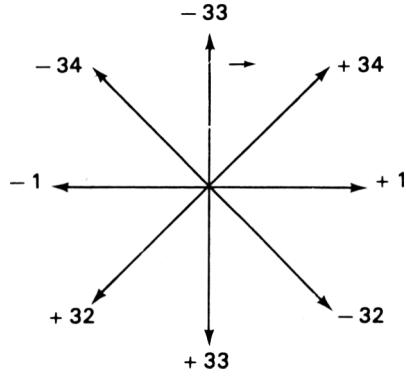
5.2.2. Applications

L'écran qui est connecté à votre ZX 81 est en réalité le reflet du contenu du buffer d'affichage. Ainsi faire se déplacer un caractère sur l'écran revient à le stocker en différents endroits du buffer d'affichage en l'effaçant de l'endroit où il se trouvait précédemment. Pour mieux "voir" les déplacements que nous allons réaliser représentons-nous le buffer d'affichage comme une superposition de 24 lignes de 33 caractères (le 33e caractère étant le caractère NEWLINE — voir schéma ci-dessous), et non comme une suite quelconque d'octets.



Nous nous apercevons ainsi que pour faire passer un caractère sur la ligne inférieure, il faudra venir le stocker 33 octets plus loin que son adresse initiale dans le buffer d'affichage.

Nous pouvons représenter les déplacements sur l'écran par le schéma suivant. Les chiffres indiqués correspondent au déplacement à effectuer dans le buffer d'affichage.



Exemple :

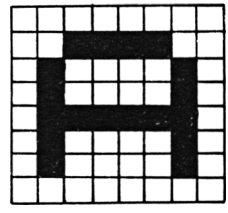
Le programme suivant affiche un caractère agrandi sur huit lignes et huit colonnes. Chargez-le à l'adresse 16675, mettez-vous en mode SLOW, lancez son exécution et appuyez sur une touche de votre choix. Si celle-ci ne correspond pas à un caractère affichable, elle sera ignorée par le programme.

LECTURE	CALL	CLAVIER	CD	BB	02	
	LD	A, L	7D			
	INC	A	3C			
	JR	Z, LECTURE	28	F9		
	PUSH	HL	E5			
	POP	BC	C1			
	CALL	CODE	CD	BD	07	
	LD	A, (HL)	7E			
	RES	7,A	CB	BF		
	CP	40 H	FE	40		
	JR	NC, LECTURE	30	ED		
	LD	A,(HL)	7E			
	LD	L, A	6F			
	LD	H,00	26	00		
	ADD	HL, HL	29			
	ADD	HL, HL	29			
	ADD	HL, HL	29			
	LD	BC,7680	01	00	1E	
	ADD	HL, BC	09			
	LD	DE,(D-FILE)	ED	5B	0C	40
	EX	DE, HL	EB			
	LD	BC,0010	01	0A	00	
	ADD	HL, BC	09			
	LD	C, 8	0E	08		
CONTINUE	LD	A, (DE)	1A			
	LD	B, 8	06	08		
BSUIVANT	RLA		17			
	JR	NC,SUITE	30	02		
	LD	(HL), 128	36	80		
SUITE	INC	HL	23			
	DJNZ	B SUIVANT	10	F8		
	PUSH	BC	C5			
	LD	BC, 25	01	19	00	
	ADD	HL, BC	09			
	INC	DE	13			
	POP	BC	C1			
	DEC	C	0D			
	JR	NZ, CONTINUE	20	EB		
	RET		C9			

Avec ce programme nous utilisons le générateur de caractères du ZX 81 situé dans la dernière partie de la ROM aux adresses 7680 à 8191. Chaque caractère y occupe huit octets dont la représentation binaire mémorise la forme du caractère. Prenons l'exemple de la lettre A. Sa représentation binaire dans le générateur de caractères est :

```
0 0 0 0 0 0 0 0
0 0 1 1 1 1 0 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 1 1 1 1 1 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0
```

ce qui donne sur l'écran

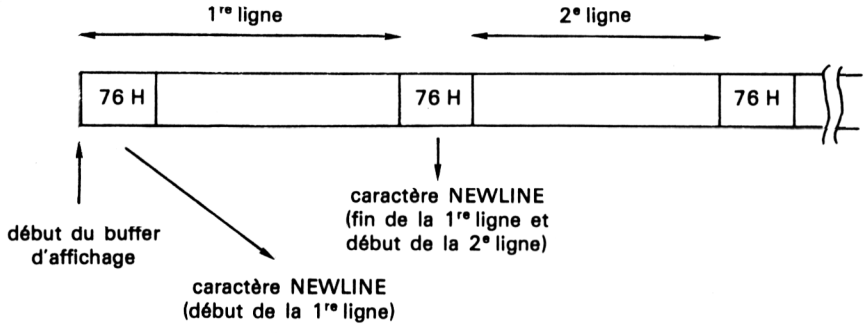


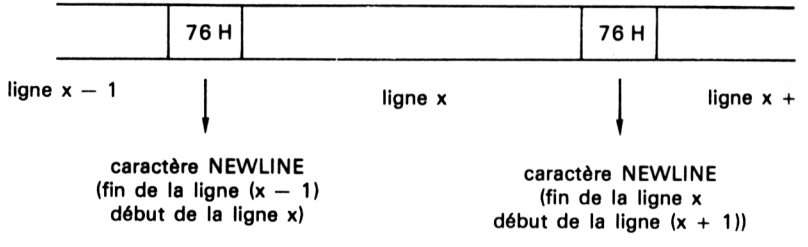
Chaque bit à 1 correspond à un point à noircir et chaque bit à zéro un point à mettre en blanc. Ainsi dans le programme, nous testons, en réalisant des décalages à gauche, chacun des 8 bits des 8 octets en écrivant le caractère ■ (espace en vidéo inversé) dans le buffer d'affichage chaque fois que nous détectons un bit à 1. L'adresse du caractère dans le générateur est obtenue en multipliant son code par huit (c'est le rôle des trois instructions ADD HL, HL consécutives) ajoutée à l'adresse de base 7680. Dès que nous aurons parcouru le premier octet, nous devrons, avant de passer au second, nous recadrer dans le buffer d'affichage sur la ligne suivante. Or, à ce moment-là, nous avons progressé par rapport au début de l'exécution, de huit octets dans le buffer d'affichage. Nous ajouterons donc 25 au contenu de la paire de registre HL qui mémorise notre position dans le buffer.

5.2.3. Utiliser le buffer d'affichage comme mémoire

Reprenons l'exemple que nous avions choisi au chapitre 4 pour illustrer la scrutation du clavier. Dans cet exemple, nous déplacions un curseur sur la première ligne de l'écran. Pour s'assurer que nous restions bien sur cette ligne, nous entretenions un compteur dont la valeur nous renseignait sur notre position — zéro signifiait que nous étions en bout de ligne, 31 en début de ligne, et une autre valeur que nous étions entre les

deux. En réalité, nous pouvons procéder plus simplement ; en effet dans le buffer d'affichage, chaque ligne est délimitée par un caractère NEWLINE (voir schéma ci-dessous) il suffit donc avant de se déplacer de venir tester si le caractère suivant est un caractère NEWLINE. Si c'est le cas nous resterons au même endroit.





Le programme deviendra alors :

	LD	HL, (D-FILE)	2A	0C	40
	INC	HL	23		
	LD	(HL), 176	36	B0	
LECTURE	LD	BC, F7FE H	01	FE	F7
	IN	A, (C)	ED	78	
	AND	10 H	E6	10	
	JR	Z, GAUCHE	28	20	
	LD	B, EF H	06	EF	
	IN	A, (C)	ED	78	
	LD	E, A	5F		
	AND	04 H	E6	04	
	JR	Z, DROITE	28	06	
	LD	A, E	7B		
	AND	01 H	E6	01	
	JR	NZ, LECTURE	20	E9	
	RET		C9		
DROITE	LD	(HL), 128	36	80	
	INC	HL	23		
	LD	A, (HL)	7E		
	CP	76 H	FE	76	
	JR	Z, FIN LIGNE	28	04	
	LD	(HL), 176	36	B0	
	JR	LECTURE	18	DC	
FIN LIGNE	DEC	HL	2B		
	LD	(HL), 176	36	B0	
	JR	LECTURE	18	D7	
GAUCHE	LD	(HL), 00	36	00	
	DEC	HL	2B		
	LD	A, (HL)	7E		
	CP	76 H	FE	76	
	JR	Z, DEBUT LIGNE	28	04	
	LD	(HL), 176	36	B0	
	JR	LECTURE	18	CB	
DEBUT LIGNE	INC	HL	23		
	LD	(HL), 176	36	B0	
	JR	LECTURE	18	C6	

Remarque: Par un souci de clarté l'exemple précédent comporte certaines séquences qui se répètent. Nous aurions pu le programmer de la manière suivante:

	LD	HL, (D-FILE)	2A	0C	40
	INC	HL	23		
	LD	(HL), 176	36	B0	
LECTURE	LD	BC, F7FE H	01	FE	F7
	IN	A, (C)	ED	78	
	AND	10 H	E6	10	
	JR	Z, GAUCHE	28	17	
	LD	B, EF H	06	EF	
	IN	A, (C)	ED	78	
	RRA		1F		
	RET	NC	D0		
	AND	02 H	E6	02	
	JR	NZ, LECTURE	20	ED	
DROITE					
	LD	(HL), 128	36	80	
GAUCH 1	INC	HL	23		
	LD	A, (HL)	7E		
	CP	76 H	FE	76	
	JR	NZ, DROIT 1	20	01	
	DEC	HL	2B		
DROIT 1	LD	(HL), 176	36	B0	
	JR	LECTURE	18	E0	
GAUCHE					
	LD	(HL), 00	36	00	
	DEC	HL	2B		
	LD	A, (HL)	7E		
	CP	76 H	FE	76	
	JR	NZ, DROIT 1	20	F4	
	JR	GAUCH 1	18	EB	

Cette version occupe 16 octets de moins que la précédente.

5.2.4. Travailler dans le buffer d'affichage sans bloc extension mémoire

Lorsque l'on travaille sans le bloc extension mémoire, le buffer d'affichage est réduit à l'initialisation, à 25 caractères NEWLINE ; il est ensuite agrandi chaque fois que l'on affiche quelque chose sur l'écran et conserve ainsi cette nouvelle taille ne reprenant sa configuration minimale que lors de l'exécution d'une instruction CLS.

Pour appliquer les méthodes précédentes, nous agrandirons donc le buffer en affichant autant de caractères que le nombre d'octets dont on souhaite disposer dans celui-ci.

Exemple: Reprenons l'exemple précédent. Nous avons besoin pour permettre le déplacement du curseur, de la première ligne de l'écran c'est-à-dire que nous devons avoir entre les deux caractères NEWLINE qui délimitent la première ligne, 32 autres caractères. Nous afficherons donc dès le début du programme, 32 caractères espace. Pour cela nous rajouterons en tête du programme la séquence suivante:

	LD	B, 32	06	20	
	LD	A, 00	3E	00	
ESPACE	RST	10 H	D7		
	DJNZ	ESPACE	10	FD	
	LD	HL, (16396)	2A	0C	40

Suite du programme (2e version)

Remarques: 1. Dans les exemples précédents, tous les débranchements sont des débranchements relatifs. Vous pouvez donc recopier le programme tel quel.

2. Pour entrer ce programme réservez une instruction REM de 55 caractères et utilisez le programme de chargement du chapitre 3 en remplaçant l'instruction 9000 par:

```
9000  FOR   N = 16514 10 16568
```

Lancez ensuite l'exécution par la commande RAND USR 16514.

Supposons que nous voulions déplacer ce curseur sur la cinquième ligne de l'écran et non sur la première. Nous appelerons donc auparavant le sous-programme situé à l'adresse 08F5H qui nous permet d'afficher un caractère à un endroit quelconque de l'écran. Toutefois, il faudra recalculer l'adresse du curseur dans le buffer d'affichage. En effet, lorsque nous travaillons sur la première ligne de l'écran, le curseur était le second caractère du buffer d'affichage (le premier étant un caractère NEWLINE), il suffisait pour connaître son adresse d'ajouter 1 à l'adresse de début du buffer. Voyons une méthode permettant de faire ce calcul simplement. La variable système DF-CC mémorise l'adresse à laquelle le prochain caractère sera stocké dans le buffer d'affichage. Après avoir affiché les 32 espaces de la cinquième ligne nous pointerons donc sur le caractère NEWLINE qui termine cette cinquième ligne. Pour placer le curseur au début de la ligne, il suffira alors de décrémenter le contenu de cette variable de 32. Le début du programme deviendra ainsi :

	LD	BC, 0004	01	00	04
	CALL	08F5 H	CD	F5	08
	LD	B, 32	06	20	
	LD	A, 00	3E	00	
ESPACE	RST	10 H	D7		
	DJNZ	ESPACE	10	FD	
	LD	HL, (16398)	2A	0E	40
	LD	DE, – 32	11	E0	FF
	ADD	HL, DE	19		
	LD	(HL), 176	36	B0	

Suite du programme

Chargez ce programme de la même manière que précédemment ; votre instruction REM devra contenir 65 caractères.

Remarque : Il était aussi possible de remplir les cinq premières lignes de l'écran avec des espaces mais les quatre premières lignes étant inutilisées, nous aurions perdu de la place en mémoire ; de plus ceci ne nous aurait pas dispensé du calcul de l'adresse du curseur.

6

Faites vos jeux !...

Remarque préliminaire: Pour tous les jeux faisant intervenir des graphiques, il faudra faire fonctionner le ZX 81 dans le mode SLOW ; en effet, dans le mode FAST l'affichage n'a lieu que lorsque le programme est terminé, ce qui pour un jeu des envahisseurs, par exemple, peut-être l'origine d'une certaine gêne !...

Un des attraits principaux de la programmation en langage machine est la possibilité d'obtenir des graphiques animés relativement rapides. Avant d'aborder l'étude détaillée de deux jeux dans lesquels interviennent des graphiques animés, revoyons quelques méthodes couramment employées dans la réalisation de jeux à base de graphiques.

6.1. QUELQUES MÉTHODES GRAPHIQUES

Un des soucis constants des applications graphiques est de ne pas " sortir " de la zone mémoire allouée au buffer d'affichage. Vous risqueriez en effet de venir écraser la zone des variables ou même votre programme !...

6.1.1. Se limiter dans le buffer d'affichage

Nous avons vu dans le chapitre précédent qu'il était relativement aisé de limiter ses déplacements sur une même ligne de l'écran, grâce aux caractères NEWLINE qui la bornent. Pourquoi ne pas appliquer ce principe à tout le buffer d'affichage ? En effet, nous pourrions remplir la première et la dernière ligne de l'écran par un caractère spécial qu'il suffirait de venir tester (comme pour le caractère NEWLINE) afin de savoir si nous avons atteint le haut ou le bas de l'écran.

Exemple :

Avec le programme suivant, nous remplirons la première ligne de l'écran avec le caractère ▀ et la dernière ligne avec le caractère ▄ (inversion vidéo du précédent). Utilisez les touches 5, 6, 7 et 8 pour déplacer le curseur respectivement à gauche, en bas, en haut ou à droite. Un appui sur la touche 0 (zéro) arrêtera le programme. Chargez ce programme avec le programme d'aide à la mise au point (commande S). Pour le lancer, utilisez la commande Basic RAND USR 16675.

DEBUT	LD	HL, (D-FILE)	2A	0C	40
	INC	HL	23		
	LD	B, 32	06	20	
	LD	A, 83	3E	83	
LIGNE 1	RST	10 H	D7		
	DJNZ	LIGNE 1	10	FD	
	LD	DE, 726	11	D6	02
	ADD	HL, DE	19		
	LD	(DF-CC), HL	22	0E	40
	LD	B, 20	06	20	
	LD	A, 03	3E	03	
LIGNE 24	RST	10 H	D7		
	DJNZ	LIGNE 24	10	FD	
	DEC	HL	2B		
	DEC	HL	2B		
SUITE	LD	(16507), HL	22	7B	40
	LD	(HL), 176	36	B0	
LECTURE	CALL	CLAVIER	CD	BB	02
	LD	A, L	7D		
	INC	A	3C		
	JR	Z, LECTURE	28	F9	

	PUSH	HL	E5		
	POP	BC	C1		
	CALL	CODE	CD	BD	07
	LD	A, (HL)	7E		
	LD	HL, (16507)	2A	7B	40
	LD	(HL), 00	36	00	
	CP	33	FE	21	
	JR	Z, GAUCHE	28	11	
	CP	34	FE	22	
	JR	Z, BAS	28	28	
	CP	35	FE	23	
	JR	Z, HAUT	28	19	
	CP	36	FE	24	
	JR	Z, DROITE	28	0D	
	CP	28	FE	1C	
	RET	Z	C8		
	JR	LECTURE	18	D9	
GAUCHE	DEC	HL	2B		
	LD	A, (HL)	7E		
	CP	118	FE	76	
	JR	Z, DROITE	28	02	
SUITE 1	JR	SUITE	18	CC	
DROITE	INC	HL	23		
	LD	A, (HL)	7E		
	CP	118	FE	76	
	JR	Z, GAUCHE	28	F2	
	JR	SUITE 1	18	F6	
HAUT	LD	DE, – 33	11	DF	FF
	ADD	HL, DE	19		
	LD	A, (HL)	7E		
	CP	131	FE	83	
	JR	Z, BAS	28	02	
SUITE 2	JR	SUITE 1	18	EB	
BAS	LD	DE, + 33	11	21	00
	ADD	HL, DE	19		
	LD	A, (HL)	7E		
	CP	03	FE	03	
	JR	Z, HAUT	28	EC	
	JR	SUITE 2	18	F3	

Chaque fois que nous nous déplaçons verticalement, nous venons tester si le caractère situé juste au-dessus ou au-dessous n'est pas le caractère ▄ (code 131) ou ▀ (code 03). Si cela est le cas, le déplacement n'est pas effectué. Pour les déplacements horizontaux, nous testerons le caractère NEWLINE.

6.1.2. Créer un mouvement

Cette méthode est habituelle. Pour déplacer une image, il suffit de la dessiner à un endroit de l'écran, puis de l'effacer pour la redessiner un peu plus loin, de l'effacer à nouveau et ainsi de suite.

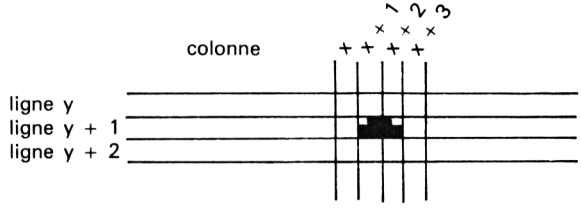
Pour éviter d'obtenir un mouvement saccadé, n'effectuez que des déplacements " pas à pas" (d'une ligne à la ligne suivante ou d'une colonne à la colonne suivante). Vous ne serez pas alors obligés d'effacer le sujet avant de le redessiner. En effet en entourant ce dessin de caractères espaces, il effacera de lui-même toute trace de son passage à la position précédente.

Exemple: Le petit programme suivant déplace diagonalement un " vaisseau spatial ".

	LD	C, 18	0E	12	
	LD	HL, (D-FILE)	2A	0C	40
	INC	HL	23		
SUITE	LD	(DF-CC), HL	22	0E	40
	PUSH	BC	C5		
	LD	B, 4	06	04	

ESPACE 1	LD	(HL), 00	36	00	
	INC	HL	23		
	DJNZ	ESPACE 1	10	FB	
	LD	DE, 0029	11	1D	00
	ADD	HL, DE	19		
	LD	(HL), 00	36	00	
	INC	HL	23		
	LD	(HL), 129	36	81	
	INC	HL	23		
	LD	(HL), 130	36	82	
	INC	HL	23		
	LD	(HL), 00	36	00	
	INC	HL	23		
	ADD	HL, DE	19		
	LD	B, 4	06	04	
ESPACE 2	LD	(HL), 00	36	00	
	INC	HL	23		
	DJNZ	ESPACE 2	10	FB	
	LD	HL, (DF-CC)	2A	0E	40
	LD	DE, 0034	11	22	00
	ADD	HL, DE	19		
	LD	BC, 1000 H	01	00	10
TEMPO	DEC	BC	0B		
	LD	A, B	78		
	OR	C	B1		
	JR	NZ, TEMPO	20	FB	
	POP	BC	C1		
	DEC	C	0D		
	JR	NZ, SUITE	20	CA	
	RET		C9		

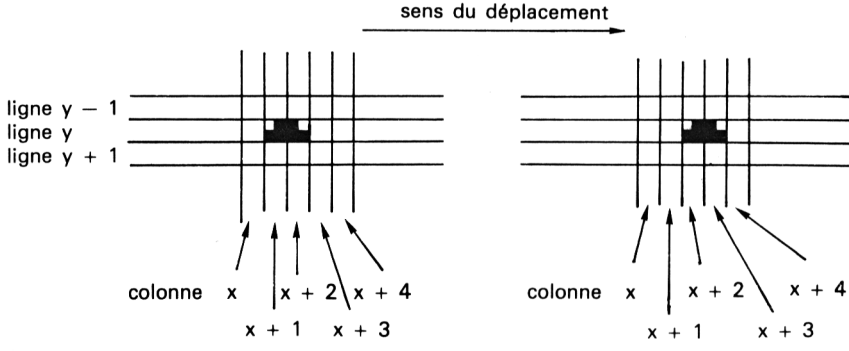
le vaisseau spatial aura l'aspect suivant :



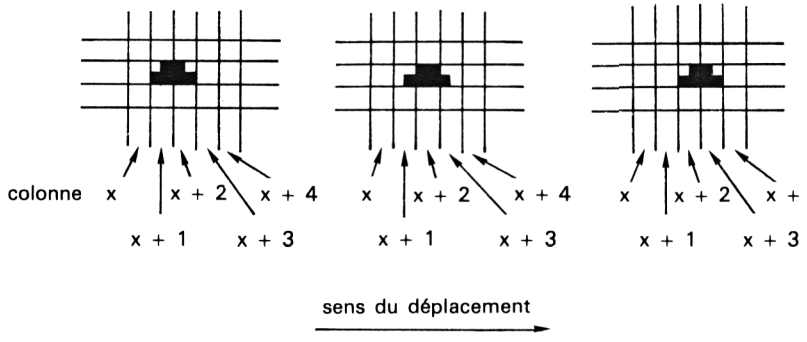
Sa position sera mémorisée dans la variable système DF-CC qui contiendra alors l'adresse du premier caractère espace de la ligne y.

6.1.3. Décomposer le mouvement

Bien souvent les déplacements ne s'effectuent que dans une direction (horizontale ou verticale). Si le sujet à déplacer s'y prête, on pourra décomposer son mouvement en plusieurs phases de manière à obtenir un déplacement plus régulier qui sera plus agréable à regarder. Reprenons le dessin du vaisseau spatial de l'exemple précédent et faisons-le se déplacer horizontalement de gauche à droite colonne par colonne. Nous pouvons représenter son déplacement par le dessin suivant :



Cependant nous pouvons aussi nous déplacer demi-colonne par demi-colonne, comme indiqué par le schéma ci-dessous :



Ceci nous oblige toutefois à utiliser plusieurs types de caractères graphiques pour représenter le même dessin, ce qui n'était pas le cas avec la méthode précédente.

Exemple: Dans cet exemple, nous déplacerons deux vaisseaux spatiaux en utilisant les deux méthodes. Le vaisseau spatial de la première ligne est déplacé demi-colonne par demi-colonne, celui de la seconde ligne colonne par colonne.

Vous remarquerez, avec cet exemple, qu'il n'est pas difficile de donner l'impression d'un déplacement *simultané* de plusieurs objets.

	LD	C, 28	0E	1C	
	LD	HL, (D-FILE)	2A	0C	40
SUITE	INC	HL	23		
	LD	(DF-CC), HL	22	0E	40
	LD	DE, 0033	11	21	00
	ADD	HL, DE	19		
	ADD	HL, DE	19		
	LD	(16507), HL	22	7B	40
	PUSH	BC	C5		
	LD	HL, (DF-CC)	2A	0E	40
	LD	(HL), 00	36	00	
	INC	HL	23		
	LD	(HL), 129	36	81	
	INC	HL	23		
	LD	(HL), 130	36	82	
	LD	BC, 0800 H	01	00	08
TEMPO 1	DEC	BC	0B		
	LD	A, B	78		
	OR	C	B1		
	JR	NZ, TEMPO 1	20	FB	
	DEC	HL	2B		
	LD	(HL), 135	36	87	
	INC	HL	23		
	LD	(HL), 128	36	80	
	INC	HL	23		
	LD	(HL), 4	36	04	
	LD	BC, 0800 H	01	00	08

TEMPOZ	DEC	BC	0B		
	LD	A, B	78		
	OR	C	B1		
	JR	NZ, TEMPOZ	20	FB	
	LD	HL, (16507)	2A	7B	40
	LD	(HL), 00	36	00	
	INC	HL	23		
	LD	(HL), 129	36	81	
	INC	HL	23		
	LD	(HL), 130	36	82	
	LD	HL, (DF-CC)	2A	0E	40
	POP	BC	C1		
	DEC	C	0D		
	JR	NZ, SUITE	20	BD	
	RET		C9		

La position de chaque vaisseau est repérée par le caractère espace le plus à gauche (ie celui qui correspond à la colonne x). Celle du premier vaisseau est mémorisée dans la variable système DF-CC, celle du second aux adresses 16507 et 16508 inutilisées par le ZX 81.

6.1.4. Utiliser des temporisations

Les vitesses de déplacement obtenues pour des graphiques réalisés avec des programmes BASIC, sont relativement lentes. Celles obtenues à partir de programmes écrits en langage machine sont généralement trop rapides !... (Cela dépend évidemment du temps pris par les traitements effectués entre deux affichages). Pour ralentir cette vitesse nous passerons du temps à ne rien faire, c'est un des rôles de la temporisation.

Exemple : Reprenez l'exemple du paragraphe 1.3. et remplacez l'instruction LD BC, 0800 H par :

```
LD BC, 0001 H          01  01  00
```

ce qui revient à annuler les temporisations donc à augmenter considérablement la vitesse de déplacement des deux vaisseaux spatiaux. Relancez l'exécution du programme et regardez bien votre écran si vous espérez voir le déplacement !...

Ainsi nous pourrons, en cours d'exécution, diminuer ou augmenter la vitesse de déplacement de certains jeux.

Exemple: Dans cet exemple nous faisons atterrir un vaisseau spatial "en douceur".

INITD	LD	HL, (D-FILE)	2A	0C	40	
	INC	HL	23			
	LD	(DF-CC), HL	22	0E	40	
	LD	DE, 0033	11	21	00	
	LD	B, 23	06	17		
BASECRAN	ADD	HL, DE	19			
	DJNZ	BAS ECRAN	10	FD		
	LD	(HL), 10	36	0A		
	INC	HL	23			
	LD	(HL), 10	36	0A		
	INC	HL	23			
	LD	(HL), 10	36	0A		
	INC	HL	23			
	LD	(HL), 10	36	0A		
	LD	BC, 0800 H	01	00	08	
INITF	LD	(16507), BC	ED	43	7B	40
	LD	B, 22	06	16		
SUITE	PUSH	BC	C5			
	LD	HL, (DF-CC)	2A	0E	40	
	LD	(HL), 00	36	00		
	INC	HL	23			
	LD	(HL), 00	36	00		
	INC	HL	23			
	LD	(HL), 00	36	00		
	DEC	HL	2B			
	DEC	HL	2B			
	ADD	HL, DE	19			
	LD	(DF-CC), HL	22	0E	40	
	LD	(HL), 00	36	00		
	INC	HL	23			
	LD	(HL), 129	36	81		
	INC	HL	23			
	LD	(HL), 130	36	82		
PLUS D	LD	BC, (16507)	ED	4B	7B	40
	INC	B	04			

PLUS F	LD	(16507), BC	ED	43	7B	40
	DEC	B	05			
TEMPO	DEC	BC	0B			
	LD	A, C	79			
	OR	B	B0			
	JR	NZ, TEMPO	20	FB		
	POP	BC	C1			
	DEC	B	05			
	JR	NZ, SUITE	20	D3		
	RET		C9			

Pour obtenir cet effet, nous augmentons tout simplement la durée de la temporisation au fur et à mesure que nous descendons. (Séquence d'instructions PLUSD à PLUSF). La séquence d'instructions INITD à INITF sert à dessiner la plate forme d'atterrissage et à initialiser la durée de la première temporisation. Le reste du programme gère le déplacement du vaisseau spatial.

6.1.5. Tirer sur une cible mouvante

Nous savons maintenant déplacer un ou plusieurs objets sur l'écran. Nous pourrions donc faire un programme pour tirer sur une cible mouvante, le tir du missile étant simulé par un appui touche sur le clavier (touche F par exemple). Le principe du programme serait alors le suivant :

* déplacer la cible,
* scruter le clavier,
* générer le tir si appui sur la touche F,
* retourner au déplacement de la cible.

Mais comment détecter si la cible a été touchée ou pas par le missile ? Vous vous souvenez que pour limiter nos déplacements sur sur une même ligne nous regardions si la position mémoire dans laquelle nous voulions ranger le curseur contenait un caractère NEWLINE, afin de savoir si nous étions arrivés en bout de ligne. (Revoyez l'exemple du chapitre 5 § 5.2.3). Nous adopterons ici la même méthode : avant de

déplacer le missile nous regarderons si la position ou nous voulons l'afficher contient un espace. Si ce n'est pas le cas, nous avons probablement atteint la cible ! Il ne nous reste plus qu'à effectuer les traitements correspondants (incrémentation d'un score, explosion de la cible, etc, etc...). Le traitement des déplacements de la cible et du missile deviendra alors :

* déplacer le missile,
* déplacer la cible,
* calculer la prochaine position du missile,
* la position calculée contient-elle un espace ?
  — Oui : retourner au déplacement du missile,
  — Non : la cible est atteinte, suite du traitement.

Exemple : Dans ce programme nous déplacerons un vaisseau spatial sur lequel vous tirez en appuyant sur la touche F. Votre canon est schématisé par le caractère C en inversion vidéo.

	LD	HL, (D-FILE)	2A	0C	40
	LD	DE, 182	11	B6	00
	ADD	HL, DE	19		
	LD	(HL), 168	36	A8	
INIT	LD	B, 32	06	20	
	LD	A, 22	3E	16	
LIMITE	RST	10 H	D7		
	DJNZ	LIMITE	10	FD	
	LD	HL, + 1	21	01	00
	LD	(SENS), HL	22	D6	41
	LD	A, 28	3E	1C	
	LD	(COMPTEUR), A	32	D8	41
	LD	HL, 00	21	00	00
	LD	(POSMIS), HL	22	D9	41
	LD	HL, (D-FILE)	2A	0C	40
	LD	DE, 34	11	22	00
	ADD	HL, DE	19		
	LD	(DF-CC), HL	22	0E	40

Dans cette séquence de programme nous limitons le haut de l'écran avec une ligne en pointillé et nous initialisons quelques variables pour la suite du programme. La position du missile est mémorisée dans la

variable POSMIS, celle du vaisseau spatial dans la variable système DF-CC. La variable SENS nous indiquera le sens du déplacement – 1, nous nous déplacerons de droite à gauche, + 1 de gauche à droite. Voyons maintenant la séquence de déplacement du vaisseau spatial.

	LD	HL, (DF-CC)	2A	0E	40	
VAISSEAU	LD	BC, (SENS)	ED	4B	D6	41
	ADD	HL, BC	09			
	LD	(DF-CC), HL	22	0E	40	
	LD	(HL), 00	36	00		
	INC	HL	23			
	LD	(HL), 129	36	81		
	INC	HL	23			
	LD	(HL), 130	36	82		
	INC	HL	23			
	LD	(HL), 00	36	00		
	LD	A, (COMPTEUR)	3A	D8	41	
	DEC	A	3D			
	LD	(COMPTEUR), A	32	D8	41	
	JR	NZ, MISSILE	20	1A		
	LD	HL, (DF-CC)	2A	OE	40	
	INC	HL	23			
	INC	B	04			
	LD	BC, – 1	01	FF	FF	
	JR	NZ, MEMOSENS	20	04		
	DEC	HL	2B			
	LD	BC, + 1	01	01	00	
MEMOSENS	LD	(SENS), BC	ED	43	D6	41
	LD	(DF-CC), HL	22	0E	40	
	LD	A, 28	3E	1C		
	LD	(COMPTEUR), A	32	D8	41	

Vous remarquerez que nous utilisons un compteur pour s'assurer que nous restons sur la même ligne plutôt que de tester le caractère NEWLINE. En effet, pour rechercher le caractère NEWLINE, il aurait d'abord fallu tester le sens afin de savoir où le chercher (à droite ou à gauche de l'envahisseur); ce qui serait plus compliqué. Avec le compteur, nous savons automatiquement que nous sommes à un des bouts de ligne et qu'il faut inverser le sens de déplacement chaque fois qu'il arrive à zéro.

Dans la séquence suivante nous scrutons le clavier et nous générons le tir du missile si l'utilisateur appui sur la touche F.

MISSILE	LD	HL, (POSMIS)	2A	D9	41
	LD	A, H	7C		
	OR	L	B5		
	JR	NZ, DEPLACEMENT	20	1A	
	CALL	CLAVIER	CD	BB	02
	LD	A, L	7D		
	INC	A	3C		
	JR	Z, TEMPO 1	28	2C	
	PUSH	HL	E5		
	POP	BC	C1		
	CALL	CODE	CD	BD	07
	LD	A, (HL)	7E		
	CP	43	FE	2B	
	JR	NZ, TEMPO1	20	22	
	LD	HL, (D-FILE)	2A	OC	40
	LD	DE, 182	11	B6	00
	ADD	HL, DE	19		
	JR	DEP 1	18	02	
DEPLA-					
CEMENT	LD	(HL), 00	36	00	
DEP 1	LD	DE, – 33	11	DF	FF
	ADD	HL, DE	19		
	LD	A, (HL)	7E		
	CP	00	FE	00	
	JR	NZ, SUITE	20	04	
	LD	(HL), 11	36	0B	
	JR	TEMPO	18	07	
SUITE	CP	22	FE	16	
	JR	NZ, TOUCHE	20	10	
	LD	HL, 00	21	00	00
TEMPO	LD	(POSMIS), HL	22	D9	41
TEMPO 1	LD	BC, 1200 H	01	00	12
TEMPO 2	DEC	BC	0B		
	LD	A, C	79		
	OR	B	B0		
	JR	NZ, TEMPO2	20	FB	
	JR	VAISSEAU	18	83	
TOUCHE	DEC	HL	2B		
	LD	(HL), 61	36	3D	
	INC	HL	23		
	LD	(HL), 61	36	3D	
	INC	HL	23		
	LD	(HL), 61	36	3D	
	RET		C9		

SENS	DEFW	0	00	00
COMPTEUR	DEFB	0	00	
POSMIS	DEFB	0	00	00

Dans cet exemple le missile se déplace à la même vitesse que la cible. Essayer d'augmenter la vitesse de celui-ci (en le déplaçant deux lignes par deux lignes, par exemple ...). Quelque fois vous verrez la moitié du vaisseau spatial disparaître et pourtant aucune explosion !... Trouvez-en la raison et modifiez le programme en conséquence !...

Pour lancer l'exécution de ce programme, utiliser la commande Basic RAND USR 16675.

6.1.6. Afficher un score

Bien sûr ! Vous désirez comptabiliser le nombre de vaisseaux spatiaux que vous avez touchés. Il faut donc gérer le score. Ceci n'est pas toujours très aisé surtout si l'on desire pouvoir l'incrémenter de plusieurs points à la fois. La méthode consistera alors à reconvertir ce score en décimal puis à afficher successivement le chiffre des unités, des dizaines, des centaines, etc...

Exemple : Le petit programme suivant affichera un score compris entre 00 et 99. Ce score pourra être incrémenté de 15 au maximum chaque fois. Pour nous éviter une conversion hexadécimale/décimale, nous supposerons que ce score est mémorisé sous sa forme décimale en utilisant l'instruction DAA.

LD	A, (16507)	3A	7B	40
LD	B, A	47		
LD	A, (16508)	3A	7C	40
ADD	A, B	80		
DAA		27		
PUSH	AF	F5		
SRL	A	CB	3F	
SRL	A	CB	3F	
SRL	A	CB	3F	
SRL	A	CB	3F	
LD	HL, (D-FILE)	2A	0C	40
INC	HL	23		

ADD	A, 28	C6	1C	
LD	(HL), A	77		
INC	HL	23		
POP	AF	F1		
AND	OF H	E6	0F	
ADD	A, 28	C6	1C	
LD	(HL), A	77		
RET		C9		

Rentrez les deux valeurs aux adresses 16507 et 16508 puis lancez l'exécution par la commande G (adresse début du programme: 16675, adr fin: 0 — pas de points d'arrêts). Prenons comme valeur 10 et 0F; le résultat affiché est 25 (ce résultat est correct compte tenu du fait que nous travaillons en décimal).

Si ce score ne doit être incrémenté que de un à chaque fois la méthode suivante est beaucoup plus simple et permet de plus, d'afficher un score aussi "grand" que l'on veut. Cette méthode, comme quelques autres déjà rencontrées, fait appel au fait que le buffer d'affichage peut être utilisé comme mémoire. Dans le programme suivant appuyer sur une des touches du clavier pour que le score s'incrémente. Pour arrêter le programme appuyez sur la touche SHIFT et une autre touche.

LECTURE	CALL	CLAVIER	CD	BB	02
	LD	A, L	7D		
	INC	A	3C		
	JR	Z, LECTURE	28	F9	
	LD	HL, (D-FILE)	2A	0C	40
	LD	DE, 30	11	1E	00
	ADD	HL, DE	19		
COMPTE	LD	A, (HL)	7E		
	CP	00	FE	00	
	JR	NZ, PLUS	20	02	
	LD	A, 28	3E	1C	
PLUS	INC	A	3C		
	CP	38	FE	26	
	JR	C, FIN	38	05	
	LD	(HL), 28	36	1C	
	DEC	HL	2B		
	JR	COMPTE	18	EF	
FIN	LD	(HL), A	77		
	CALL	CLAVIER	CD	BB	02
	RR	H	CB	1C	
	RET	NC	D0		
	JR	LECTURE	18	D8	

Regardez comment fonctionne ce petit programme. Comme indications rappelez-vous que le code 28 représente le chiffre 0, le code 38 le chiffre 10 et que après une comparaison le flag Report (Carry) est positionné à 1 si le contenu de l'accumulateur est inférieur à l'octet avec lequel on le compare. Si on désire mémoriser le score il sera plus simple de le faire dans une variable à part et d'incrémenter cette variable à chaque fois.

6.1.7. Jeu du mur de briques

Appliquons ces différentes méthodes au jeu du mur de briques. Dans ce jeu vous disposez de trois balles pour détruire un mur de 80 briques. Pour bouger la raquette utilisez les touches 1 (gauche) et 0 (droite). La balle sera représentée par la lettre 0. Ce jeu a été décomposé en plusieurs sous-programmes, dont certains (et même tous !...) sont relativement simples.

Pour charger ce programme en mémoire, procédez comme suit:

1. Charger le programme d'aide à la mise au point dans le haut de la mémoire à l'adresse 32000 (cf. chapitre 3).
2. Créer une instruction REM de 512 octets (cf. chapitre 2). Sur ces 512 octets, nous n'en utiliserons que 366. Vous disposerez donc de 146 octets pour apporter toutes les améliorations que vous jugerez nécessaires et indispensables...
3. Détruire le programme de création de l'instruction REM.
4. Charger le programme en langage machine.
5. Ajouter à votre programme BASIC les instructions suivantes.

```
10 LET D=INT (20*RND)
20 POKE 16828,D
30 RAND USR 16830
40 STOP
```

6. Entrez la commande POKE 17091,255. Ceci fait disparaître toutes les lignes à partir de la ligne 8020.

Le premier sous-programme dessine un cadre de 24 lignes sur 20 colonnes. Vous remarquerez que ce cadre est fermé en bas par une ligne pointillée. Ceci nous servira à déterminer si la balle sort du jeu ou pas. Ce sous-programme assure aussi le dessin de la raquette.

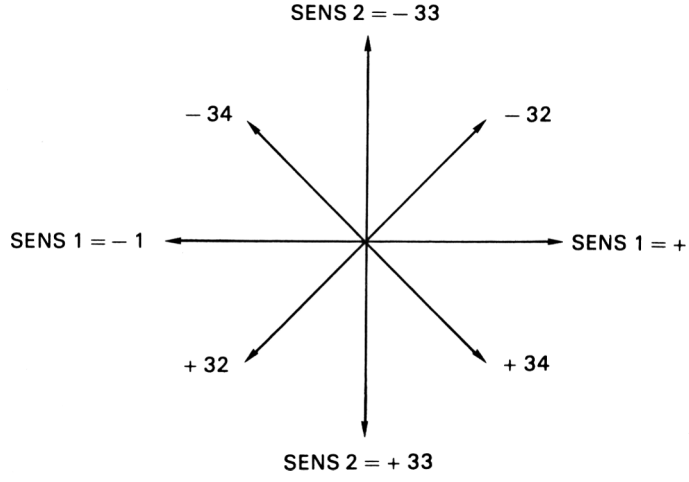
Mnémoniques			Adresses	Code hexadécimal		
CADRE	LD	HL, (D-FILE)	16514	2A	0C	40
	INC	HL	17	23		
	LD	(HL), 135	18	36	87	
	LD	B, 20	20	06	14	
CADRE 1	INC	HL	22	23		
	LD	(HL), 131	23	36	83	
	DJNZ	CADRE 1	25	10	FB	
	INC	HL	27	23		
	LD	(HL), 04	28	36	04	
	LD	B, 21	30	06	15	
CADRE 2	LD	DE, 12	32	11	0C	00
	ADD	HL, DE	35	19		
	LD	(HL), 133	36	36	85	
	LD	DE, 0021	38	11	15	00
	ADD	HL, DE	41	19		
	LD	(HL), 05	42	36	05	
	DJNZ	CADRE 2	44	10	F2	
	LD	DE, 12	46	11	0C	00
	ADD	HL, DE	49	19		
	LD	(HL), 133	50	36	85	
	LD	DE, 10	52	11	0A	00
	ADD	HL, DE	55	19		
	LD	HL), 03	56	36	03	
	INC	HL	58	23		
	LD	(HL), 03	59	36	03	
	LD	(16507), HL	61	22	7B	40
	INC	HL	64	23		
	LD	(HL), 03	65	36	0B	
	ADD	HL, DE	67	19		
	DEC	HL	68	2B		
	LD	(HL), 05	69	36	05	
	LD	DE, 0012	71	11	0C	00
	ADD	HL, DE	74	19		
	LD	B, 22	75	06	16	
CADRE 3	LD	(HL), 22	77	36	16	
	INC	HL	79	23		
	DJNZ	CADRE 3	80	10	FB	
	RET		82	C9		

Sous-programme de temporisation (Sans commentaires...).

TEMPO	DEC	BC	16583	OB	
	LD	A, C	84	79	
	OR	B	85	B0	
	JR	NZ, TEMPO	86	20	FB
	RET		88	C9	

Sous-programme d'initialisation

Dans ce sous-programme, nous initialisons la position de la balle (POS) et son sens. La position de la balle est choisie aléatoirement en utilisant le contenu de la variable RAND. Cette variable a été initialisée avec le petit programme Basic précédent. La balle se déplacera toujours en biais ; son sens est déterminé grâce aux deux variables SENS1 et SENS2 ; SENS1 mémorise les déplacements horizontaux (à savoir + 1 à droite, – 1 à gauche) et SENS2 les déplacements verticaux (à savoir + 33 vers le bas, – 33 vers le haut). La combinaison de ces deux valeurs nous donnera un déplacement en biais.



INIT B	LD	HL, (D-FILE)	16589	2A	0C	40
	LD	DE, 0332	92	11	4C	01
	ADD	HL, DE	95	19		
	LD	D, O	96	16	00	
	LD	A, (RAND)	98	3A	BC	41
	NOP		16601	00		
	NOP			00		
	LD	E, A	03	5F		
	ADD	HL, DE	04	19		
	LD	(POS), HL	05	22	ED	40
	LD	HL, – 1	08	21	FF	FF
	LD	(SENS 1), HL	11	22	EF	40
	LD	HL, + 33	14	21	21	00
	LD	(SENS 2), HL	17	22	F1	40
	RET		20	C9		
POS	DEFW	0	21	00	00	
SENS 1	DEFW	0	23	00	00	
SENS 2	DEFW	0	25	00	00	

Sous-programme de dessin du mur:

Le mur est représenté par le caractère graphique dont le code est 08. Un espace de quatre lignes est laissé entre le mur et le haut du cadre.

MUR	LD	HL, (D-FILE)	16627	2A	0C	40
	LD	DE, 166	30	11	A6	00
	ADD	HL, DE	33	19		
	LD	C, 4	34	0E	04	
MUR1	LD	B, 20	36	06	14	
	LD	DE, 0013	38	11	0D	00
MUR2	INC	HL	41	23		
	LD	(HL), 8	42	36	08	
	DJNZ	MUR2	44	10	FB	
	ADD	HL, DE	46	19		
	DEC	C	47	0D		
	JR	NZ, MUR1	48	20	F2	
	RET		50	C9		

Déplacement de la raquette

La raquette "occupe" trois colonnes sur l'écran. Elle est repérée par la position de son centre qui est mémorisé dans la variable système située à l'adresse 16507. Pour déplacer la raquette vers la droite appuyez sur la touche 0, vers la gauche sur la touche 1. Ce sous-programme est

RAQUETTE						
	LD	HL, (16507)	16651	2A	7B	40
	LD	BC, F7FE H	54	01	FE	F7
	IN	A, (C)	57	ED	78	
	RRA		59	1F		
	JR	NC, GAUCHE	60	30	08	
	LD	B, EF H	62	06	EF	
	IN	A, (C)	64	ED	78	
	RRA		66	1F		
	JR	NC, DROITE	67	30	12	
	RET		69	C9		
GAUCHE						
	DEC	HL	70	2B		
	DEC	HL	71	2B		
	LD	A, (HL)	72	7E		
	CP	133	73	FE	85	
	RET	Z	75	C8		
	LD	(HL), 03	76	36	03	
	INC	HL	78	23		
	LD	(16507), HL	79	22	7B	40
	INC	HL	82	23		
	INC	HL		23		
	LD	(HL), 00	84	36	00	
	RET		86	C9		
DROITE	INC	HL	87	23		
	INC	HL	88	23		
	LD	A, (HL)	89	7E		
	CP	05	90	FE	05	
	RET	Z	92	C8		
	LD	(HL), 03	93	36	03	
	DEC	HL	95	2B		
	LD	HL	96	22	7B	40
	DEC	HL	99	2B		
	DEC	HL	16700	2B		
	LD	(HL), 00	01	36	00	
	RET		03	C9		

l'application directe de la méthode étudiée au chapitre 4 — Scrutation du clavier. En effet, nous testerons si nous avons atteint un des côtés du cadre pour limiter le déplacement.

Déplacement de la balle

Cette partie est un peu plus longue que les autres! La balle se déplacera toujours en biais, pour cela nous utiliserons les deux variables SENS1 et SENS2 — voir partie Initialisation. La position de la balle est mémorisée dans la variable POS.

Le déplacement de la balle est réalisée de la façon suivante:

1. La balle n'a rien heurté. On s'assure d'abord que la nouvelle position de la balle est occupée par un espace. Si ce n'est pas le cas, on considérera que la balle a heurté quelque chose et on effectuera le traitement correspondant; sinon on place la balle à cet endroit et on retourne au programme principal.

2. La balle a heurté un objet. Cela peut être:

— *un des côtés du cadre*: faire rebondir la balle en inversant la variable SENS 1,

— *la raquette ou le haut du cadre*: faire rebondir la balle en inversant la variable SENS 2,

— *le mur*: incrémenter le score, enlever du mur la brique touchée, en la remplaçant par un caractère blanc, faire rebondir la balle en inversant la variable SENS 2,

— *la ligne pointillée dans le bas du cadre*: mettre le flag Z à 1 et retourner au programme principal.

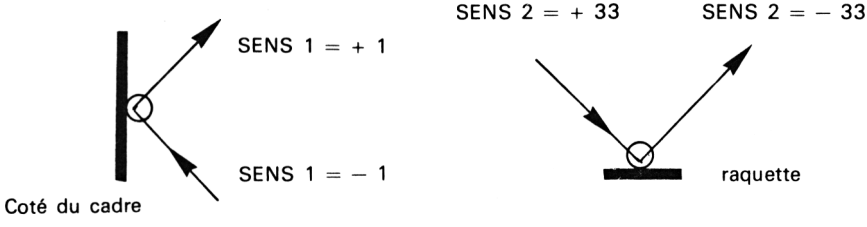


Schéma des déplacements

COTE	INC	B	16778	04		
	JR	Z, COTE 2	79	28	05	
	LD	BC, – 1	81	01	FF	FF
	JR	COTE 3	84	18	02	
COTE 2	LD	C, 1	86	0E	01	
COTE 3	ADD	HL, BC	88	09		
	JR	FIN MOUV	89	18	0B	
HAUT	INC	D	91	14		
	JR	Z, HAUT 2	92	28	05	
	LD	DE, – 33	94	11	DF	FF
	JR	HAUT 3	97	18	02	
HAUT 2	LD	E, + 33	99	1E	21	
HAUT 3	ADD	HL, DE	801	19		
FIN MOUV	ADD	HL, BC	02	09		
	LD	(SENS 1), BC	03	ED	43	EF 40
	LD	(SENS 2), DE	07	ED	53	F1 40
	JR	POSITION	11	18	A0	
MP1	ADD	HL, BC	813	09		
	LD	A, (HL)	14	7E		
	CP	00	15	FE	00	
	JP	NZ, FMP2	17	C2	4F	41
	LD	HL, 34 H	20	36	34	
	LD	(POS), HL	22	22	ED	40
	JP	FMP1	25	C3	86	41
RAND	DEF B	00	16828	00		
NOMBRE	DEF B	00	29	00		

Le traitement MP1 est une correction apportée au programme initial.

Après avoir déterminé le sens de la balle, nous calculerons sa nouvelle position en nous assurant que celle-ci est libre. Si ce n'est pas le cas, nous réeffectuerons le traitement 2. Ceci explique pourquoi plusieurs briques peuvent disparaître simultanément.

BALLE			16704				
	LD	HL, (POS)	704	2A	ED	40	
	LD	BC, (SENS 1)	707	ED	4B	EF	40
	LD	DE, (SENS 2)	711	ED	5B	F1	40
	LD	(HL), 00		36	00		
POSITION	ADD	HL, DE	717	19			
	LD	A, (HL)	18	7E			
FMP2	CP	22	19	FE	16		
	RET	Z	21	C8			
	CP	08	22	FE	08		
	JR	NZ, SUITE	24	20	21		
	LD	(HL), 00	26	36	00		
	PUSH	HL	28	E5			
	PUSH	DE	29	D5			
	LD	HL, (D-FILE)	30	2A	0C	40	
	LD	DE, 97	33	11	61	00	
	ADD	HL, DE	36	19			
COMPTE	LD	A, (HL)	37	7E			
	CP	00	38	FE	00		
	JR	NZ, PLUS	40	20	02		
	LD	A, 28	42	3E	1C		
PLUS	INC	A	44	3C			
	CP	38	45	FE	26		
	JR	C, FIN COMPTE	47	38	05		
	LD	(HL), 28	49	36	1C		
	DEC	HL	51	2B		C3	
	JR	COMPTE	52	18	EF	F1	41
FINCOMPTE	LD	(HL), A	54	77			
	POP	DE	55	D1			
	POP	HL	56	E1			
	JR	HAUT	57	18	20		
SUITE	RES	Z, A	16759	CB	BF		
	CP	5	61	FE	05		
	JR	Z, COTE	63	28	0D		
	CP	3	65	FE	03		
	JR	Z, HAUT	67	28	16		
	JP	MP1	69	C3	AD	41	
	NOP		72	00			
	NOP		73	00			

FMP 1	LD	A, 1	74	3E	01	
	OR	A	76	B7		
	RET		77	C9		

La dernière partie est le programme principal dans lequel nous appelons tous les sous-programmes. A chaque passage, le programme de déplacement de la raquette est appelé deux fois pour vous permettre d'aller un peu plus vite que la balle.

La position initiale de chaque balle est modifiée en appliquant l'instruction SRL A au contenu de la variable RAND (Ceci revient en réalité à diviser le contenu de cette variable par 2).

DEBUT	LD	A, 03	16830	3E	03	
	LD	(NOMBRE), A	32	32	BD	41
	CALL	CADRE	35	CD	82	40
	CALL	MUR	38	CD	F3	40
INIT	CALL	INIT B	41	CD	CD	40
BOUCLE	LD	BC, 1800 H	44	01	00	18
	CALL	TEMPO	47	CD	C7	40
	CALL	RAQUETTE	50	CD	0B	41
	CALL	BALLE	53	CD	40	41
	JR	NZ, CONTINUE	56	20	12	
	LD	A, (NOMBRE)	58	3A	BD	41
	DEC	A	61	3D		
	RET	Z	62	C8		
	LD	(NOMBRE), A	63	32	BD	41
	LD	A, (RAND)	66	3A	BC	41
	SRL	A	69	CB	3E	
	LD	(RAND), A	71	32	BC	41
	JR	INIT	16	18	DD	
CONTINUE	CALL	RAQUETTE	76	CD	0B	41
	JR	BOUCLE	79	18	DB	

Plusieurs améliorations peuvent être apportées à ce programme. La modification suivante vous permettra d'augmenter la vitesse du jeu.

Introduisez les trois octets suivants à l'adresse 16751.

JP	16881	16751	C3	F1	41

et rajoutez la séquence ci-dessous à l'adresse 16881.

LD	A, (16846)	16881	3A	CE	41
CP	08	84	FE	08	
JR	Z, CONT	86	28	04	
DEC	A	88	3D		
LD	(16846), A	89	32	CE	41
DEC	HL	92	2B		
JP	COMPTE	93	C3	61	41

Dans cette version de programme, nous ne testons pas si le mur de briques est complètement détruit. Apportez une modification pour en tenir compte.

De plus il n'y a aucune temporisation entre chaque balle. Rajoutez-la !

Jeux des envahisseurs

Vous trouverez dans le livre " ZX 81 à la conquête des jeux ", paru dans la même collection, un jeu des envahisseurs réalisé en langage machine. Je ne vous donnerai pas le listage de ce jeu ici. Reportez-vous au livre cité. Nous en détaillerons cependant quelques points.

Ce programme comporte trois sous-programmes, un sous-programme d'initialisation situé à l'adresse 16528, un sous-programme de temporisation situé à l'adresse 16539, un sous-programme de dessin des envahisseurs situé à l'adresse 16571.

L'état de chaque envahisseur est mémorisé dans une table. Pour un envahisseur, l'octet correspondant est initialisé à — 1 s'il est toujours présent et à zéro s'il a été abattu. L'organigramme du programme principal est le suivant :

— Scruter le clavier.

— Si appui sur la touche 0 aller au traitement de tir du défenseur.

— Sinon gérer le curseur selon les autres appuis touche.

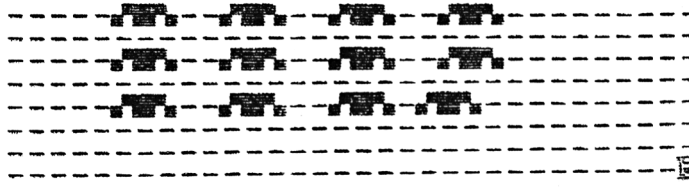
— Dessiner un envahisseur sur les douze.

— Faire tirer cet envahisseur.

— L'envahisseur a-t-il touché le défenseur ?

Oui : fin du programme.
Non : dessiner l'envahisseur suivant.

Le sous-programme d'initialisation remplit la table des envahisseurs à FF — il est extrêmement simple. Le programme de dessin de l'envahisseur est détaillé ci-après car il comporte quelques points intéressants, notamment le calcul de la position de chaque envahisseur. Les autres parties du programme ne posent pas de difficultés particulières. Reportez-vous au livre "ZX 81 à la conquête des jeux" pour voir comment elles sont programmées. L'aspect du jeu est le suivant, il vous permettra de mieux comprendre le sous-programme.



Dessin d'un envahisseur

A chaque passage dans le programme, un envahisseur est dessiné. Le numéro de l'envahisseur à dessiner est mémorisé dans la variable NUMERO. La séquence suivante incrémente le numéro de l'envahisseur à dessiner et test s'il n'a pas été abattu (auquel cas il ne faut pas le dessiner).

```
ENV     LD      A, (NUMERO)
        INC     A
        LD      E, A
        LD      D, 0
        LD      HL, TABLE ENVAHISSEUR
        ADD     HL, DE
        LD      (NUMERO), A
        LD      A, (HL)
        OR      A
        JP      Z, TIR
```

l'envahisseur dont le numéro est dans le registre E doit être dessiné, il faut calculer son adresse dans le buffer d'affichage. Un envahisseur ayant l'aspect suivant: il occupera donc cinq octets.

en tenant compte des caractères "tirets" qui l'entourent). De plus il y a

quatre envahisseurs par ligne:

```
            | LD   |  HL, (D-FILE)
            | LD   |  BC, 0005
NOUV-LIGNE  | LD   |  A, 04
CALCUL      | DEC  |  E
            | JR   |  Z, ADR CALCULEE
            | ADD  |  HL, BC
            | DEC  |  A
            | JR   |  NZ, CALCUL
```

Si l'on arrive à la séquence suivante, c'est que l'envahisseur à dessiner ne se trouve pas sur la ligne en cours d'exploration. Il faut donc passer à la ligne d'envahisseur suivante. Sur la ligne en cours d'exploration, nous avons progressé de 20 caractères (4 × 5. Il faut donc ajouter 46 pour se situer au début de la ligne d'envahisseurs suivante. (13 pour arriver au bout de la ligne en cours + 33 pour tenir compte de la ligne pointillée entre chaque ligne d'envahisseurs).

```
PUSH | DE          |
LD   | DE, 0046    |
ADD  | HL, DE      |
POP  | DE          |
JR   | NOUV-LIGNE  |
```

L'adresse à laquelle il faut dessiner l'envahisseur est presque connu. En effet les envahisseurs "balayant" une ligne complète, il faut tenir compte du décalage par rapport au début de la ligne; celui-ci est mémorisé dans la variable DEPLACEMENT. En effet, le dessin est effectué selon la méthode suivante:

Dessiner les douze envahisseurs.

Tester si on est arrivé au bout de la ligne?

Oui: Changer de sens et retourner au dessin des envahisseurs.

Non: Décaler de une colonne dans le sens sélectionné et retourner au dessin des envahisseurs.

Le traitement est:

```
ADR CALCULEE
            LD      DE, (DEPLACEMENT)
            LD      D, 0
            ADD     HL, DE
            PUSH    HL
            LD      A, (NUMERO)
            SUB     12
            JP      M, DESSIN
            LD      A, 0
            LD      (NUMERO),A
            LD      HL, DEPLACEMENT
            LD      B, (HL)
            LD      A, (SENS)
            ADD     A, B
            LD      (DEPLACEMENT), A
```

Cette séquence teste s'il faut inverser le sens ou pas. On ne peut faire que 13 déplacements dans un sens ou dans l'autre.

```
            LD      B, 1
            LD      HL, SENS
            CP      B
            JR      NZ, A2
            LD      (HL), 1
A2          LD      B, 13
            CP      B
            JR      NZ, A1
            LD      (HL), FF
A1          POP     HL
            JMP     ENV
```

la variable SENS indique le sens du déplacement + 1 on se déplace vers la droite, — 1 vers la gauche.

Dessin d'un envahisseur. La position de celui-ci est mémorisée par son point central dans la variable POS ayant 16507 comme adresse.

```
DESSIN   LD    DE, (SENS)
         LD    D, 00
         INC   E
         DEC   DE
         ADD   HL, DE
         LD    (HL), 22
         INC   HL
         LD    (HL), 06
         INC   HL
         LD    (HL), 128
         LD    (POS), HL
         INC   HL
         LD    (HL), 134
         INC   HL
         LD    (HL), 22
```

Vous pouvez remarquer la combinaison astucieuse des deux instructions :

```
INC    E
DEC    DE
```

Ceci est utile pour le cas ou la variable SENS contient la valeur – 1 le registre E contient donc FF. Le fait de l'incrémenter le fait devenir nul mais en refaisant une décrémentation sur la *paire* de registres DE, la valeur négavite se "propage" aussi dans le registre D. Ainsi si le contenu de la paire de registres DE était 00FF il deviendra FFFF et pourra à ce moment-là être utilisé pour le calcul de l'adresse avec la paire de registres HL.

Dans ce jeu des envahisseurs deux autres points sont intéressants.

1. L'utilisation de l'instruction basic :

```
PRINT  AT  7,31 + 0 * USR 16528;
```

Pourquoi cette instruction pour aller exécuter le programme d'initialisation qui se contente simplement de remplir la table des envahisseurs avec la valeur – 1 (pour les déclarer présents) ? Ceci sert à initialiser la variable système (DF-CC) avec l'adresse du dernier caractère de la huitième ligne. Cette variable sera utilisée pour mémoriser la position du curseur : 6 en inversion vidéo.

2. L'exécution du programme de dessin d'un envahisseur est lancée par l'instruction :

```
60  GOTO  USR 16799
```

En rentrant dans le détail du programme, on s'aperçoit de la chose suivante.

Si le défenseur est touché la paire de registres BC est initialisée à 00, sinon elle est initialisée à 60. L'instruction suivante est une instruction RET. Donc l'instruction 60 sera équivalente à l'instruction.

```
60  GOTO  00        si le défenseur est touché
60  GOTO  60        si le défenseur n'est pas touché
```

Dans le premier cas le programme recommencera au début, et dans le second cas l'envahisseur suivant sera dessiné. Cette méthode procure en réalité l'avantage suivant : pouvoir utiliser la touche BREAK comme dans un programme BASIC normal !

7

Utiliser un assembleur

Jusqu'à présent pour charger un programme en langage machine, nous utilisions le programme d'aide à la mise au point, qui nous permettait de rentrer directement en hexadécimal les codes opérations de chaque instruction. Ceci est déjà une amélioration si on établit une comparaison avec la méthode faisant appel à des instructions POKE successives. Cette méthode possède toutefois plusieurs désavantages :

— C'est une méthode longue, il faut convertir les mnémoniques du langage assembleur en hexadécimal.

— Le risque d'erreur est important notamment dans les calculs d'adresse (débranchements relatifs entre autres...).

— La correction des programmes est mal aisée et ne peut se faire que par "rapiéçages". Voir exemple du mur de briques — ce qui contribue à augmenter considérablement la taille du programme en langage machine.

Pour éviter ces différents inconvénients, nous pouvons utiliser un assembleur qui se chargera alors de convertir les mnémoniques, de calculer les adresses, etc... En résumé, il s'occupe du travail fastidieux. Voyons comment l'employer. L'assembleur présenté est ZX AS développé par la société BUG-BYTE.

1. Charger le programme assembleur à partir de la cassette.

2. Faites RUN. Le programme est alors chargé dans le haut de la mémoire et occupe les cinq derniers kilo-octets (adresses 27648 à

32767). Vous pouvez alors supprimer les instructions 10, 20 et 30, mais attention: enlevez d'abord l'instruction 10; n'essayez pas de détruire d'abord l'instruction 20, cela pourrait vous ennuyer quelque peu...

3. Vous pouvez maintenant rentrer votre programme en incluant les mnémoniques dans des instruction REM. Tous les mnémoniques du Z 80 sont autorisés avec la différence suivante, la virgule est remplacée par un point.

Ainsi le mnémonique Z 80: LD A, B devra être écrit LD A.B pour être compris par l'assembleur ZX AS. Pensez-y! Ceci est, avec un caractère espace oublié, une des causes fréquentes de l'erreur 4. Le programme devra être compris entre parenthèses celles-ci étant inclues dans les instructions REM.

Exemple: Nous reprenons comme exemple le petit programme du chapitre 2 avec lequel nous tracions une diagonale sur l'écran. Son listage est le suivant:

```
   1 REM XXXXXXXXXXXXXXX
  10 REM (
  20 REM LD DE.0034
  30 REM LD HL.(16396)
  40 REM INC HL
  50 REM LD B.20
  60 REM :L0LD (HL).$86
  70 REM ADD HL.DE
  80 REM DJNZ.L0
  90 REM RET
 100 REM )
9000 FAST
9010 INPUT ZZZ
9020 POKE 32641,INT (ZZZ/256)
9030 POKE 32640,ZZZ-256*INT (ZZZ
/256)
9040 RAND USR 28565
9050 PRINT AT 21,0;"ERROR ";PEEK
 32651
9060 SLOW
```

l'instruction 1 REM xx.....xx sert à réserver de la place en mémoire. Quand nous lancerons l'assemblage, le programme ZX AS nous demandera l'adresse à laquelle le résultat devra être chargé, il suffira de répondre 16514 pour que notre programme en langage machine soit chargé à cette adresse.

Les autres instructions REM contiennent le programme. Une petite restriction concernant les étiquettes: Vous ne pouvez pas prendre

n'importe quel nom. En effet, une étiquette sera définie par les deux caractères : L suivi d'un numéro compris entre 0 et 255 — voir instruction 60.

Une quantité hexadécimale sera indiquée en utilisant le caractère **$** — voir instruction 60.

Vous trouvez ci-dessous le résultat de l'assemblage tel qu'il apparaît sur votre écran. Le report ERROR 0 signifie qu'il n'y a aucune erreur.

```
0020 4082 11 22 00      LD DE.0034
0030 4085 2A 0C 40      LD HL.(163
96)
0040 4088 23            INC HL
0050 4089 06 14         LD B.20
0060 408B 36 86         LD (HL).$8
6
0070 408D 19            ADD HL.DE
0080 408E 10 00         DJNZ.L0
0090 4090 C9            RET
0020 4082 11 22 00      LD DE.0034
0030 4085 2A 0C 40      LD HL.(163
96)
0040 4088 23            INC HL
0050 4089 06 14         LD B.20
0060 408B 36 86         LD (HL).$8
6
0070 408D 19            ADD HL.DE
0080 408E 10 FB         DJNZ.L0
0090 4090 C9            RET

ERROR 0
```

Ce résultat apparaît sous la forme de quatre colonnes. La colonne de gauche vous rapppelle le numéro de l'instruction REM dans laquelle se trouve le mnémonique, la colonne suivante vous indique l'adresse hexadécimale à laquelle le code objet a été rangé (4082 H = 16514), les deux dernières colonnes vous donnent le code hexadécimal de l'instruction (11 22 00) et l'instruction elle-même (LD DE.0034).

Remarque: Il est possible d'insérer plusieurs instructions assembleur dans la même instruction REM à condition de séparer chacune d'entre elles par un point virgule.

Le caractère étoile '*' est utilisé pour l'introduction de commentaires.

Nous obtenons alors le résultat suivant:

```
    1 REM XXXXXXXXXXXXXXX
   10 REM (
   15 REM * EXEMPLE DE COMMENTAIR
E
   20 REM LD DE.0034;LD HL.(16396
);INC HL;LD B.20
   30 REM :L0 LD (HL).$86;ADD HL.
DE;DJNZ.L0:RET
  100 REM )
9000 FAST
9010 INPUT ZZZ
9020 POKE 32641,INT (ZZZ/256)
9030 POKE 32640,ZZZ-256*INT (ZZZ
/256)
9040 RAND USR 28565
9050 PRINT AT 21,0;"ERROR ";PEEK
 32651
9060 SLOW

0020 4082 11 22 00     LD DE.0034
0020 4085 2A 0C 40     LD HL.(163
96)
0020 4088 23           INC HL
0020 4089 06 14        LD B.20
0030 408B 36 86        LD (HL).$8
6
0030 408D 19           ADD HL.DE
0030 408E 10 00        DJNZ.L0:RE
T
0020 4082 11 22 00     LD DE.0034
0020 4085 2A 0C 40     LD HL.(163
96)
0020 4088 23           INC HL
0020 4089 06 14        LD B.20
0030 408B 36 86        LD (HL).$8
6
0030 408D 19           ADD HL.DE
0030 408E 10 FB        DJNZ.L0:RE
T

ERROR 0
```

ANNEXE 1
Fonctionnement du programme d'aide à la mise au point et améliorations possibles

1. PROGRAMME BASIC

Ce programme comporte certaines parties écrites en BASIC et trois sous-programmes écrits en langage machine. Examinons les parties écrites en BASIC ; celles-ci ne devraient pas vous poser trop de problèmes.

— Les instructions REM 3, 4 et 5 "contiennent" les trois sous-programmes écrits en langage machine.

— L'instruction 6 REM xxx servira à mémoriser 3 octets (voir ci-après sous-programme en langage machine pour la commande G).

— L'instruction 7 REM x.....x contient 206 caractères utiles et commence à l'adresse décimale 16675. Vous pouvez utiliser la place occupée par cette instruction REM pour stocker vos propres sous-programmes écrits en langage machine.

— Instructions 8020 à 8080 : initialisations.

— Instructions 8085 à 8115 : acquisition du caractère de commande et stockage de ce caractère à l'adresse 16444.

— Instruction 8120 : Recherche de la commande à exécuter et branchement à ce traitement.

— Instructions 8125 à 8190 : Commande D (Dump).

— Instructions 8195 à 8255 : Commande S (Substitution).

— Instructions 8260 à 8275 : Commande M (Recopie d'une zone mémoire).

— Instructions 8280 à 8295 : Commande G (Exécution d'un sous-programme écrit en langage machine).

— Instructions 8300 à 8385 : Commande R (Affichage du contenu des registres).

— Instruction 8390 : Commande F.

— Instruction 8395 à 8440 : Sous-programmes d'acquisition de trois adresses décimales au plus (mémorisées dans le tableau F) et rangement de ces adresses sous forme hexadécimale dans les six premiers octets du buffer de l'imprimante (c'est-à-dire les octets situés aux adresses 16444 à 16449).

— Instructions 8445 à 8450 : Sous-programme d'affichage d'un nombre sous sa forme hexadécimale. Le nombre à afficher doit être rangé au préalable dans la variable E.

2. SOUS-PROGRAMME ÉCRITS EN LANGAGE MACHINE

Dans ce qui suit vous trouverez le listing commenté de chaque sous-programme dont le code hexadécimal peut être retrouvé au chapitre 3.

2.1. Sous-programme de recherche de la commande

Ce sous-programme s'adresse dans une table contenant chaque commande et le numéro de l'instruction BASIC à laquelle cette commande débute dans le programme BASIC.

La commande est stockée à l'adresse décimale 16444. Ce sous-programme débute à l'adresse 16514.

RECHERCHE DE LA COMMANDE

```
DEBU1
        LD      DE, 0003
        LD      C, 06           ; (C) = nbre de commandes
        LD      HL, TCOMM       ; (HL) = adresse table des
                                  commandes
        LD      A, (16444)      ; (A) = Commande à
                                  exécuter

SUIT1
        CP      (HL)            ; est-ce cette commande ?
        JR      Z, SUITE        ; Oui va rechercher le nu-
                                  méro de l'instruction
                                  BASIC correspondante
        ADD     HL, DE
        DEC     C               ; A-t-on examiner toute la
                                  liste des commandes ?
        JR      NZ, SUIT1       ; Non passe à la commande
                                  suivante
        LD      BC, 8085        ; Oui commande inexis-
                                  tante demande
                                  ; une nouvelle commande
        RET

SUITE
        INC     HL              ; la commande a été trou-
                                  vée. Récupère
        LD      C, (HL)         ; le numéro de l'instruction
        INC     HL              ; BASIC correspondante
        LD      B, (HL)
        RET
TCOMM                           ; Table des commandes
        DEFB    29 H            ; Commande D
        DEFW    8125            ; instruction 8125
        DEFB    38 H            ; Commande S
        DEFW    8195
```

```
DEFB   32 H    ; Commande M
DEFW   8260
DEFB   2C H    ; Commande G
DEFW   8280
DEFB   37 H    ; Commande R
DEFW   8300
DEFB   2B H    ; Commande F
DEFW   8390
```

2.2. Sous-programme de recopie d'une zone mémoire

```
DEBU2
        LD     DE, (16448)   ; (DE) = adresse zone ré-
                             ceptrice
        LD     HL, (16446)   ; Calcul nombre d'octets
        LD     BC, (16444)   ; à recopier
        PUSH   BC
        AND    A             ; Remet flag report à zéro
        SBC    HL, BC
        INC    HL
        PUSH   HL
        POP    BC            ; (BC) = nbre d'octets à re-
                             copier
        POP    HL            ; (HL) = adresse zone
                             émettrice
        LDIR
        RET
```

2.3. Sous-programme d'exécution d'un programme écrit en langage machine

Ce sous programme se divise en plusieurs parties:

* Recherche si l'utilisateur a spécifié un point d'arrêt (ie deuxième adresse de la commande G égale à zéro?)

```
DEBU3
        LD      HL, 16446
        LD      A, (HL)
        INC     HL
        OR      (HL)
        LD      HL, (16444)
        JR      Z, GO          ; la deuxième adresse est
                               nulle
                               ; pas de point d'arrêt va
                               ; exécuter le programme en
                               ; langage machine de
                               l'utilisateur
```

* La séquence suivante est exécutée si l'utilisateur a spécifié un point d'arrêt. Les trois octets situés à partir de l'adresse du point d'arrêt seront remplacés par un appel au sous-programme de sauvegarde des registres.

```
        LD      (16465), SP    ; Sauvegarde du pointeur
                               de pile
        PUSH    HL
        LD      BC, 0003       ; Recopie les trois octets
                               situés à partir
        LD      HL, (16446)    ; du point d'arrêt dans
                               les trois
        PUSH    HL             ; Caractères x de l'instruc-
                               tion 6 REM xxx
        LD      DE, 16666
        LDIR
        POP     HL             ; Remplace ces 3 octerts
                               par un appel
        LD      (HL), CD H     ; au sous-programme de
        INC     HL             ; Sauvegarde (CD = Code
                               du CALL)
        LD      (H), F8 H
        INC     HL
        LD      (HL), 40 H
        POP     HL             ; (HL) = adresse de début
                               ; du sous-programme
                               ; langage machine
                               ; de l'utilisateur
GO      JP      HL             ; Exécution du sous-
                               programme utilisateur
```

* **Sous-programme de Sauvegarde des registres**

Ce sous-programme sauvegarde le contenu des registres binaires du Z 80 (sauf les registres I et R) dans une zone de buffer de l'imprimante. L'état des registres pourra être visualisé par la commande R. De plus avant de retourner au BASIC, ce sous-programme remettra à jour le contenu des trois octets situés à partir du point d'arrêt.

```
SAVE
        LD      SP, 16465       ; initialise pointeur de pile
        PUSH    IY              ; Sauvegarde des registres
        PUSH    IX
        PUSH    HL
        PUSH    DE
        PUSH    BC
        PUSH    AF
        LD      BC, 0003        ; Remise à jour des trois
                                  octets situés
        LD      HL, 16666       ; à partir du point d'arrêt
        LD      DE, (16446)
        LDIR
        LD      SP, (16465)     ; Remise à jour du pointeur
                                  de pile
```

2.4. UTILISATION DU BUFFER DE L'IMPRIMANTE

Dans cette partie nous indiquons l'utilisation de chaque octet du buffer de l'imprimante faite par le sous-programme d'aide à la mise au 16467 - 16476

Adresses	Utilisation
16444	Commande demandée
16444 - 16445	1re adresse d'une commande
16446 - 16447	2e adresse d'une commande
16448 - 16449	3e adresse d'une commande

16450	
16451	Inutilisés
16452	
16453	Registre Flag : F
16454	Accumulateur : A
16455	Registre C
16456	Registre B
16457	Registre E
16458	Registre D
16459	Registre L
16460	Registre H
16461 - 16462	Registre d'index IX
16463 - 16464	Registre d'index IY
16465 - 16466	Pointeur de Pile utilisateur
16469 - 16476	Inutilisés

3. AMÉLIORATIONS AU PROGRAMME D'AIDE A LA MISE AU POINT

Plusieurs améliorations peuvent être apportées au programme d'aide à la mise au point.

1. Sauvegarder les registres I et R et les registres secondaires.
2. Écrire ce programme complètement en langage machine.

Le premier point est extrêmement simple. Le second est un peu plus délicat... mais vous y parviendrez sûrement ! Nous pouvons cependant apporter une autre amélioration. Dans le chapitre 3, nous indiquons qu'avec la commande G il est impossible de faire repartir l'exécution du programme en langage machine de l'utilisateur à partir d'un point d'arrêt. Il faut la faire recommencer à partir du début du sous-programme. Ceci est uniquement dû au fait que les contenus des registres du Z 80 ont été détruits. En effet une fois l'exécution du programme en langage machine de l'utilisateur terminée, le programme d'aide à la mise au point exécute quelques instructions BASIC, notamment les instructions 8095 PRINT "?"; et 8100 INPUT C$ qui nous permettent de spécifier une nouvelle commande. Ces instructions correspondent en réalité à des traitements

précis en langage machine dans la ROM 8K du ZX 81. Ces traitements utilisent les registres du Z 80 et modifient donc leurs contenus.

Pour pouvoir relancer l'exécution à partir du point d'arrêt, il faudra donc réinitialiser les registres du Z 80 avec les valeurs qui avaient été sauvegardées.

Le programme suivant permet de relancer l'exécution à partir du point d'arrêt. Pour rentrer ce programme, détruisez les instructions 5 et 6 du programme d'aide à la mise au point et remplacez-les par l'instruction :

```
5    REM    XX........XX
            98 caractères
```

Commande G

Le début du programme ne change pratiquement pas. On teste si l'utilisateur a spécifié un point d'arrêt. Si c'est le cas les trois octets situés à partir du point d'arrêt sont remplacés par un appel au sous-programme de sauvegarde puis on exécutera le traitement de réinitialisation des registres. Si l'utilisateur n'a pas spécifié de point d'arrêt, ce traitement sera effectué immédiatement.

```
DEBUT
        LD      HL, 16446
        LD      A, (HL)
        INC     HL
        LD      HL, (16444)     ; (HL) = adresse début pro-
                                  gramme utilisateur
        JZ      Z, REGISTRES    ; va au traitement de réinitia-
                                  lisation des registres si
                                  pas de point d'arrêt
```

```
; point d'arrêt spécifié
|        |                  |
|        |                  |
|  LD    |  (16465), SP     |
|  PUSH  |  HL              |
|  LD    |  BC, 0003        |
|  LD    |  HL, (16446)     |
|  PUSH  |  HL              |
|  LD    |  DE, 16788       |
|  LDIR  |                  |
|  POP   |  HL              |
|  LD    |  (HL), CD H      |
|  INC   |  HL              |
|  LD    |  (HL), 14 H      |
|  INC   |  HL              |
|  LD    |  (HL), 41 H      |
|  POP   |  HL              |  ; (HL) = adresse début pro-
                                gramme utilisateur
```

Cette séquence ne comporte que des modifications d'adresse par rapport à la version proposée précédemment au paragraphe 2.3. La séquence suivante effectue la réinitialisation des registres. Remarquez que avant de commencer à exécuter cette séquence la paire de registres HL contient l'adresse à laquelle il faudra commencer l'exécution du programme en langage machine de l'utilisateur. Nous utilisons le même principe que pour le programme de sauvegarde. En effet nous viendrons écraser l'adresse 0000 de l'instruction JP 0000 H par l'adresse à laquelle l'exécution du programme de l'utilisateur doit être commencée.

```
REGISTRES
        EX      DE, HL
        LD      HL, 16658
        LD      (HL), E          ; Remplace l'adresse
        INC     HL               ; 0000H de l'instruction
        LD      (HL), D          ; JP 0000 H par l'adresse à
                                 ; laquelle il faut commencer
                                 ; l'exécution du programme
                                 ; utilisateur
        LD      (16465), SP
        LD      SP, 16453
        POP     AF
        POP     BC
        POP     DE
        POP     HL
        POP     IX
        POP     IY
        LD      SP, (16465)
GO
        JP      0000 H
```

La séquence suivante assure la sauvegarde des registres lorsque l'utilisateur a spécifié un point d'arrêt. Cette séquence ne comporte que des modifications d'adresse par rapport à la séquence proposée au paragraphe 2.3.

```
SAVE
        LD      SP, 16465
        PUSH    IY
        PUSH    IX
        PUSH    HL
        PUSH    DE
        PUSH    BC
        PUSH    AF
        LD      BC, 0003
        LD      HL, 16688
        LD      DE, (16446)
        LDIR
        LD      SP, (16465)
        RET
```

Le code hexadécimal de ce programme est le suivant:

```
?D
16593 16690

16593 21 3E 40 7E 23 B6 2A 3C
16601 40 28 1B ED 73 51 40 E5
16609 01 03 00 2A 3E 40 E5 11
16617 30 41 ED B0 E1 36 CD 23
16625 36 14 23 36 41 E1 EB 21
16633 12 41 73 23 72 ED 73 51
16641 40 31 45 40 F1 C1 D1 E1
16649 DD E1 FD E1 ED 7B 51 40
16657 C3 00 00 31 51 40 FD E5
16665 DD E5 E5 D5 C5 F5 01 03
16673 00 21 30 41 ED 56 3E 40
16681 ED B0 ED 7B 51 40 C9 3D
16689 3D 3D

?
```

Remarque: Avec cette version, il faudra **obligatoirement** travailler en mode **FAST**.

Si vous désirez travailler dans le mode SLOW, il ne faudra pas effectuer la réinitialisation du registre d'index IX. (En effet, ce registre est utilisé pour l'affichage si on réinitialise ce registre, son contenu sera forcé à zéro à la première exécution du programme — le buffer de l'imprimante étant remis à zéro par la commande RUN — et ceci fait " planter " le ZX 81). La réinitialisation du registre IY devient alors délicate, nous la supprimerons donc. Ainsi pour travailler dans le mode SLOW, il faudra remplacer le contenu des quatres octets situés aux adresses 16449, 16450, 16451 et 16452 par le code hexadécimal 00 (c'est-à-dire l'instruction NOP).

Remarque: Avec cette version le premier octet disponible de l'instruction 7 REM x.....x se trouve à l'adresse décimale 16697 au lieu de l'adresse 16675.

Exemple d'utilisation

Considérons le petit programme suivant:

LD	A, 45 H	16697	3E	45	
LD	BC, 1234 H	99	01	34	12
LD	DE, 4567 H	702	11	67	45
LD	HL, 0032 H	05	21	32	00
INC	A	08	3C		
INC	BC	09	03		
INC	HL	10	23		
INC	DE	11	13		
RET		12	C9		

Nous avons d'abord mis un point d'arrêt à l'adresse 16708 puis l'exécution a été relancée à partir de cette adresse. Vous pouvez vérifier que les registres ont bien été réinitialisés. En effet, les contenus des registres A, C, E et HL sont bien incrémentés.

```
?G
16697 16708

?R
 F=00  A=45  C=34  B=12  E=67  D
=45 HL=0032 IX=028F IY=4000


?G
16708 16712

?R
 F=00  A=46  C=35  B=12  E=68  D
=45 HL=0033 IX=028F IY=4000
```

Cette amélioration est surtout utile dans les traitements de boucle.

Exemple: Le programme suivant recherche le premier caractère 'A' dans une chaîne de 10 caractères.

	LD	B, 10	697	06	0A	
	LD	HL, CHAINE	99	21	48	41
SUITE	LD	A, (HL)	702	7E		
	CP	38	703	FE	26	
	JR	Z, FIN	05	28	03	
	INC	HL	07	23		
	DEC	B	08	05		
	JR	NZ, SUITE	09	20	F7	
FIN	RET		11	C9		
DEFM	xxxAxxxxxx		12	3D	25 3D	3D 3D
			17	3D	3D 3D	3D 3D

Vérifions que nous trouvons bien le caractère 'A' au deuxième passage dans la boucle.

```
?G
16697 16705

?R
 F=22  A=3D  C=00  B=0A  E=00  D
=00 HL=4148 IX=028F IY=4000
?G
16705 16702

?G
16702 16705

?R
 F=62  A=26  C=00  B=09  E=00  D
=00 HL=4149 IX=028F IY=4000
?
```

1er passage dans la boucle

2e passage dans la boucle

Remarques :

1. Pour effectuer le deuxième passage dans la boucle, nous sommes obligés de passer par une étape intermédiaire (G. 16705. 16702). Il n'est pas possible de faire (G. 16705. 16705).

2. Il faudra veiller à ce qu'il y ait au minimum trois octets entre l'adresse de début de l'exécution du programme en langage machine et l'adresse du point d'arrêt. Rappelez-vous en effet que les trois octets

situés à partir du point d'arrêt sont remplacés par un appel au sous-programme de sauvegarde des registres. Reportez au paragraphe 2.3. du chapitre 3.

Il est aussi particulièrement intéressant de pouvoir modifier le contenu des registres. Transformons la commande R de manière à la faire fonctionner comme la commande S (le contenu du registre suivant sera affiché par appui sur la touche NEWLINE. La commande R affichera alors au début uniquement le contenu du registre F. L'affichage du contenu des registres pourra être arrêté en rentrant un point décimal). Le programme BASIC pour la commande R devient ainsi:

```
8300 LET X=16453
8305 FOR I=1 TO 6
8310 PRINT R$(I);"=";
8315 LET E=PEEK X
8320 GOSUB 8445
8321 INPUT C$
8322 IF C$="." THEN GOTO 8085
8323 IF C$="" THEN GOTO 8329
8324 PRINT "-";C$( TO 2);
8325 POKE X,16*CODE C$+CODE C$(2
)-476
8329 LET X=X+1
8330 PRINT " ";
8335 NEXT I
8340 FOR I=7 TO  0
8345 PRINT R$(I);"=";
8350 LET E=PEEK (X+1)
8355 GOSUB 8445
8360 LET E=PEEK X
8364 GOSUB 8445
8365 INPUT C$
8366 IF C$="." THEN GOTO 8085
8367 IF C$="" THEN GOTO 8374
8368 PRINT "-";C$( TO 4);
8369 POKE (X+1),16*CODE C$+CODE
C$(2)-476
8370 POKE X,16*CODE C$(3)+CODE C
$(4)-476
8374 PRINT " ";
8375 LET X=X+2
8380 NEXT I
8385 GOTO 8085
```

Exemple d'utilisation:

Reprenons le petit programme précédent:

```
LD      A, 45 H
LD      BC, 1234 H
LD      DE, 4567 H
LD      HL, 0032 H
INC     A
INC     BC
INC     HL
INC     DE
RET
```

Lançons son exécution et forçons les contenus des registres A et D à zéro en utilisant la commande R.

```
?G
16697 16708

?R
 F=00  A=45-00  C=34  B=12  E=67
  D=45-00 HL=0032
```

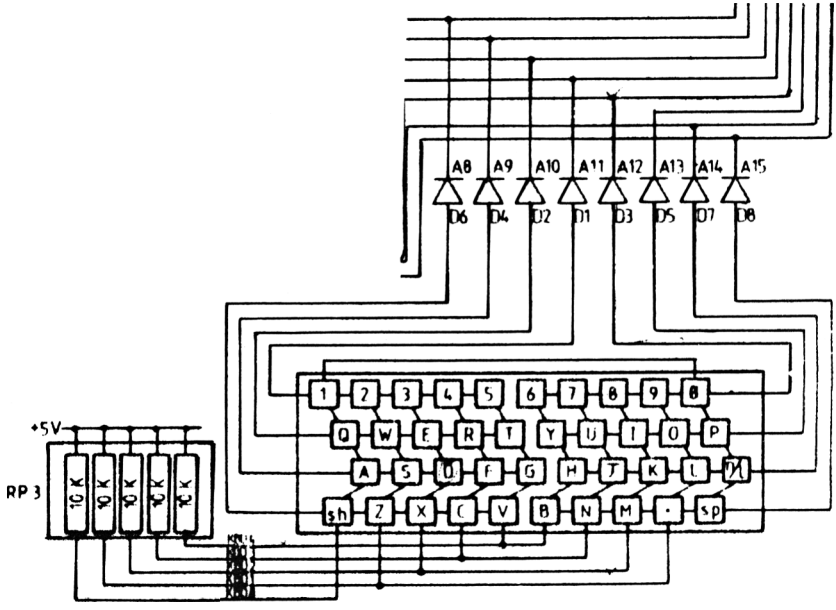
(L'affichage du contenu des registres a été arrêté après la paire de registre HL).

Relançons l'exécution pour vérifier que les registre A et D ont bien été forcés à zéro (nous devons donc obtenir A = 01 et D = 00).

```
?G
16708 16712

?R
 F=00  A=01  C=35  B=12  E=68  D
=00 HL=0033 IX=028F IY=4000 SP=7
FE0
```

ANNEXE 2

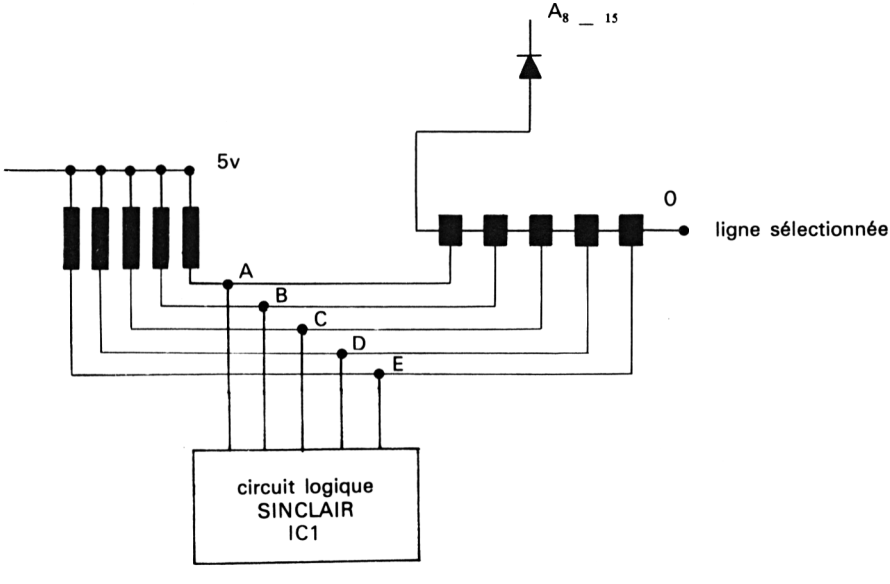
Fonctionnement du clavier du ZX 81



Une touche est un simple interrupteur entre une ligne et une colonne, ouvert au repos.

Le clavier du ZX 81 est divisé en huit lignes de cinq colonnes chacune (voir schéma). Chacune de ces huit lignes est reliée respectivement, à travers une diode, à chacun des 8 fils de poids forts du bus d'adresse. Ainsi en mettant à zéro un des fils des poids forts du bus d'adresse, nous imposons un niveau zéro sur toute la ligne correspondante.

D'autre part, la lecture de l'état des cinq touches de la ligne est effectuée aux points A, B, C, D, E (ces cinq points sont reliées au circuit logique SINCLAIR — IC1). Au repos, c'est-à-dire lorsqu'aucune touche n'est enfoncée, ces cinq points sont au niveau logique " 1 ". Lorsqu'une touche est enfoncée, elle établit un contact entre la ligne sélectionnée (au niveau logique zéro) et la colonne sur laquelle s'effectue la lecture de l'état des touches. Ce point de lecture prendra donc le niveau logique 0. Ce qui explique pourquoi une touche enfoncée a le bit qui lui correspond dans l'accumulateur forcé à zéro lors de la lecture.



Reportez-vous aussi au livre: "MICRO ORDINATEUR COMMENT ÇA MARCHE?" paru dans la même collection.

ANNEXE 3

Instructions du Z 80 et leurs codes

d : donnée sur 8 bits ou déplacement
dd : donnée sur 16 bits
aa : adresse sur 16 bits

Dans cette partie les instructions du Z 80 sont classées par ordre alphabétique.

CODE OBJET	INSTRUCTION		CODE OBJET	INSTRUCTION		CODE OBJET	INSTRUCTION	
8E	ADC	A,(HL)	C6d	ADD	A,d	DDCBd46	BIT	0,(IX+d)
DD8Ed	ADC	A,(IX+d)	09	ADD	HL,BC	FDCBd46	BIT	0,(IY+d)
FD8Ed	ADC	A,(IY+d)	19	ADD	HL,DE	CB47	BIT	0,A
8F	ADC	A,A	29	ADD	HL,HL	CB40	BIT	0,B
88	ADC	A,B	39	ADD	HL,SP	CB41	BIT	O,C
89	ADC	A,C	DD09	ADD	IX,BC	CB42	BIT	0,D
8A	ADC	A,D	DD19	ADD	IX,DE	CB43	BIT	0,E
8B	ADC	A,E	DD29	ADD	IX,IX	CB44	BIT	0,H
8C	ADC	A,H	DD39	ADD	IX,SP	CB45	BIT	0,L
8D	ADC	A,L	D09	ADD	IY,BC	CB4E	BIT	1 (HL)
CEd	ADC	A,d	FD19	ADD	IY,DE	DDCBd4E	BIT	1,(IX+d)
ED4A	ADC	HL,BC	FD29	ADD	IY,IY	FDCBd4E	BIT	1,(IY+d)
ED5A	ADC	HL,DE	FD39	ADD	IY,SP	CB4F	BIT	1,A
ED6A	ADC	HL,HL	A6	AND	(HL)	CB48	BIT	1,B
ED7A	ADC	HL,SP	DDA6d	AND	(IX+d)	CB49	BIT	1,C
86	ADD	A,(HL)	FDA6d	AND	(IY+d)	CB4A	BIT	1,D
DD86d	ADD	A,(IX+d)	A7	AND	A	CB4B	BIT	1,E
FD86d	ADD	A,(IY+d)	A0	AND	B	CB4C	BIT	1,H
87	ADD	A,A	A1	AND	C	CB4D	BIT	1,L
80	ADD	A,B	A2	AND	D	CB56	BIT	2,(HL)
81	ADD	A,C	A3	AND	E	DDCBd56	BIT	2,(IX+d)
82	ADD	A,D	A4	AND	H	FDCBd56	BIT	2,(IY+d)
83	ADD	A,E	A5	AND	L	CB57	BIT	2,A
84	ADD	A,H	E6d	AND	d	CB50	BIT	2,B
85	ADD	A,L	CB46	BIT	0,(HL)	CB51	BIT	2,C

CODE OBJET	INSTRUCTION		CODE OBJET	INSTRUCTION		CODE OBJET	INSTRUCTION	
CB52	BIT	2,D	F4dd	CALL	P,dd	34	INC	(HL)
CB53	BIT	2,E	ECdd	CALL	PE,dd	DD34d	INC	(IX+d)
CB54	BIT	2,H	E4dd	CALL	PO,dd	FD34d	INC	(IY+d)
CB55	BIT	2,L	CCdd	CALL	Z,dd	3C	INC	A
CB5E	BIT	3,(HL)	CDdd	CALL	dd	04	INC	B
DDCBd5E	BIT	3,(IX+d)	3F	CCF		03	INC	BC
FDCBd5E	BIT	3,(IY+d)	BE	CP	(HL)	0C	INC	C
CB5F	BIT	3,A	DDBEd	CP	(IX+d)	14	INC	D
CB58	BIT	3,B	FDBEd	CP	(IY+d)	13	INC	DE
CB59	BIT	3,C	BF	CP	A	1C	INC	E
CB5A	BIT	3,D	B8	CP	B	24	INC	H
CB5B	BIT	3,E	B9	CP	C	23	INC	HL
CB5C	BIT	3,H	BA	CP	D	DD23	INC	IX
CB5D	BIT	3,L	BB	CP	E	FD23	INC	IY
CB66	BIT	4,(HL)	BC	CP	H	2C	INC	L
DDCBd66	BIT	4,(IX+d)	BD	CP	L	33	INC	SP
FDCBd66	BIT	4,(IY+d)	FEd	CP	d	DBd	IN	A,(d)
CB67	BIT	4,A	EDA9	CPD		EDAA	IND	
CB60	BIT	4,B	EDB9	CPDR		EDBA	INDR	
CB61	BIT	4,C	ED81	CPIR		EDA2	INI	
CB62	BIT	4,D	EDA1	CPI		EDB2	INIR	
CB63	BIT	4,E	2F	CPL		C3dd	JP	dd
CB64	BIT	4,H	27	DAA		E9	JP	(HL)
CB65	BIT	4,L	35	DEC	(HL)	DDE9	JP	(IX)
CB6E	BIT	5,(HL)	DD35d	DEC	(IX+d)	FDE9	JP	(IY)
DDCBd6E	BIT	5,(IX+d)	FD35d	DEC	(IY+d)	DAdd	JP	C,aa
FDCBd6E	BIT	5,(IY+d)	3D	DEC	A	FAaa	JP	M,aa
CB6F	BIT	5,A	05	DEC	B	D2aa	JP	NC,aa
CB68	BIT	5,B	0B	DEC	BC	C2aa	JP	NZ,aa
CB69	BIT	5,C	0D	DEC	C	F2aa	JP	P,aa
C86A	BIT	5,D	15	DEC	D	EAaa	JP	PE,aa
CB6B	BIT	5,E	1B	DEC	DE	E2aa	JP	PO,aa
CB6C	BIT	5,H	1D	DEC	E	CAaa	JP	Z,aa
CB6D	BIT	5,L	25	DEC	H	38d	JR	C,d
DB76	BIT	6,(HL)	28	DEC	HL	30d	JR	NC,d
DDCBd76	BIT	6,(IX+d)	DD2B	DEC	IX	20d	JR	NZ,d
FDCBd76	BIT	6,(IY+d)	FD2B	DEC	IY	28d	JR	Z,d
CB77	BIT	6,A	2D	DEC	L	18d	JR	d
CB70	BIT	6,B	3B	DEC	SP	02	LD	(BC),A
CB71	BIT	6,C	F3	DI		12	LD	(DE),A
CB72	BIT	6,D	10d	DJNZ	d	77	LD	(HL),A
			FB	EI		70	LD	(HL),B
CB73	BIT	6,E	E3	EX	(SP),HL	71	LD	(HL),C
CB74	BIT	6,H	DDE3	EX	(SP),IX	72	LD	(HL),D
CB75	BIT	6,L	FDE3	EX	(SP),IY	73	LD	(HL),E
CB7E	BIT	7,(HL)	08	EX	AF,AF'	74	LD	(HL),H
DDCBd7E	BIT	7,(IX+d)	EB	EX	DE,HL	75	LD	(HL),L
FDCBd7E	BIT	7,(IY+d)	D9	EXX		36d	LD	(HL),d
CB7F	BIT	7,A	76	HALT		DD77d	LD	(IX+d),A
CB78	BIT	7,B	ED46	IM	0	DD70d	LD	(IY+d),B
CB79	BIT	7,C	ED56	IM	1	DD71d	LD	(IX+d),C
CB7A	BIT	7,D	ED5E	IM	2	DD72d	LD	(IX+d),D
CB7B	BIT	7,E	ED78	IN	A,(C)	DD73d	LD	(IX+d),E
CB7C	BIT	7,H	ED40	IN	B,(C)	DD74d	LD	(IX+d),H
CB7D	BIT	7,L	ED48	IN	C,(C)	DD75d	LD	(IX+d),L
DCaa	CALL	C,aa	ED50	IN	D,(C)	DD36d20	LD	(IX+d),d
FCaa	CALL	M,aa	ED58	IN	E,(C)	FD77d5	LD	(IY+d),A
D4aa	CALL	NC,aa	ED60	IN	H,(C)	FD70d	LD	(IY+d),B
C4aa	CALL	NZ,aa	ED68	IN	L,(C)	FD71d	LD	(IY+d),C

CODE OBJET	INSTRUCTION		CODE OBJET	INSTRUCTION		CODE OBJET	INSTRUCTION	
FD72d	LD	(IY+d),D	52	LD	D,D	00	NOP	DDCBd96
FD73d	LD	(IY+d),E	53	LD	D,E	B6	OR	(HL)
FD74d	LD	(IY+d),H	54	LD	D,H	DDB6d	OR	(IX+d)
FD75d	LD	(IY+d),L	55	LD	D,L	FDB6d	OR	(IY+d)
FD36d20	LD	(IY+d),d	16	LD	D,d	B7	OR	A
32dd	LD	(dd),A	ED5Bdd	LD	DE,(dd)	B0	OR	B
ED43dd	LD	(dd),BC	11dd	LD	DE,dd	B1	OR	C
ED53dd	LD	(dd),DE	5E	LD	E,(HL)	B2	OR	D
22dd	LD	(dd),HL	DD5Ed	LD	E,(IX+d)	B3	OR	E
DD22dd	LD	(dd),IX	FD5Ed	LD	E,(IY+d)	B4	OR	H
FD22dd	LD	(dd),IY	5F	LD	E,A	B5	OR	L
ED73dd	LD	(dd),SP	58	LD	E,B	F6d	OR	d
0A	LD	A,(BC)	59	LD	E,C	ED8B	OTDR	
1A	LD	A,(DE)	5A	LD	E,D	EDB3	OTIR	
7E	LD	A,(HL)	5B	LD	E,E	ED79	OUT	(C),A
DD7Ed	LD	A,(IX+d)	5C	LD	E,H	ED41	OUT	(C),B
FD7Ed	LD	A,(IY+d)	5D	LD	E,L	ED49	OUT	(C),C
3Add	LD	A,(dd)	1E20	LD	E,n	ED51	OUT	(C),D
7F	LD	A,A	66	LD	H,(HL)	ED59	OUT	(C),E
78	LD	A,B	DD66d	LD	H,(IX+d)	ED61	OUT	(C),H
79	LD	A,C	FD66d	LD	H,(IY+d)	ED69	OUT	(C),L
7A	LD	A,D	67	LD	H,A	D3d	OUT	(d),A
7B	LD	A,E	60	LD	H,B	EDAB	OUTD	
7C	LD	A,H	61	LD	H,C	EDA3	OUTI	
ED57	LD	A,I	62	LD	H,D	F1	POP	AF
7D	LD	AL	63	LD	H,E	C1	POP	BC
3E	LD	A,d	64	LD	H,H	D1	POP	DE
ED5F	LD	A,R	65	LD	H,L	E1	POP	HL
46	LD	B,(HL)	26d	LD	H,d	DDE1	POP	IX
DD46d	LD	B,(IX+d)	2Add	LD	HL,(dd)	FDE1	POP	IY
FD46d	LD	B,(IY+d)	21dd	LD	HL,dd	F5	PUSH	AF
47	LD	B,A	ED47	LD	I,A	C5	PUSH	BC
40	LD	B,B	DD2Add	LD	IX,(dd)	D5	PUSH	DE
41	LD	B,C	DD21dd	LD	IX,dd	E5	PUSH	HL
42	LD	B,D	FD2Add	LD	IY,(dd)	DDE5	PUSH	IX
43	LD	B,E	FD21dd	LD	IY,dd	FDE5	PUSH	IY
44	LD	B,H	6E	LD	L,(HL)	CB86	RES	0,(HL)
45	LD	B,L	DD6Ed	LD	L,(IX+d)	DDCBd86	RES	0,(IX+d)
06d	LD	B,d	FD6Ed	LD	L,(IY+d)	FDCBd86	RES	0,(IY+d)
ED4Bdd	LD	BC,(dd)	6F	LD	L,A	DB87	RES	0,A
01dd	LD	NC,dd	68	LD	L,B	DB80	RES	0,B
4E	LD	C,(HL)	69	LD	L,C	CB81	RES	0,C
DD4Ed	LD	C,(IX+d)	6A	LD	L,D	CB82	RES	0,D
FD4Ed	LD	C,(IY+d)	6B	LD	L,E	CB83	RES	0,E
4F	LD	C,A	6C	LD	L,H	CB84	RES	0,H
48	LD	C,B	6D	LD	L,L	CB85	RES	0,L
49	LD	C,C	2Ed	LD	L,d	CB8E	RES	1,(HL)
4A	LD	C,D	ED4F	LD	R,A	DDCBd8E	RES	1,(IX+d)
4B	LD	C,E	ED7Bdd	LD	SP,(dd)	FDCBd8E	RES	1,(IY+d)
4C	LD	C,H	F9	LD	SP,HL	CB8F	RES	1,A
4D	LD	C,L	DDF9	LD	SP,IX	CB88	RES	1,B
0Ed	LD	C,d	FDF9	LD	SP,IY	CB89	RES	1,C
56	LD	D,(HL)	31dd	LD	SP,dd	CB8A	RES	1,D
DD56d	LD	D,(IX+d)	EDA8	LDD		CB8B	RES	1,E
FD56d	LD	D,(IY+d)	EDB8	LDDR		CB8C	RES	1,H
57	LD	D,A	EDA0	LDI		CB8D	RES	1,L
50	LD	D,B	EDB0	LDIR		CB96	RES	2,(HL)
51	LD	D,C	ED44	NEF		DDCBd96	RES	2,(IX+d)

CODE OBJET	INSTRUCTION		CODE OBJET	INSTRUCTION		CODE OBJET	INSTRUCTION	
FDCBd96	RES	2,(IY+d)	CBBD	RES	7,L	C7	RST	00H
CB97	RES	2,A	C9	RET		CF	RST	08H
CB90	RES	2,B	D8	RET	C	D7	RST	10H
CB91	RES	2,C	F8	RET	M	DF	RST	18H
CB92	RES	2,D	D0	RET	NC	E7	RST	20H
CB93	RES	2,E	C0	RET	NZ	EF	RST	28H
CB94	RES	2,H	F0	RET	P	F7	RST	30H
CB95	RES	2,L	E8	RET	PE	FF	RST	38H
CB9E	RES	3,(HL)	E0	RET	PO	DEd	SBC	A,d
DDCBd9E	RES	3,(IX+d)	C8	RET	Z	9E	SBC	A,(HL)
FDCBd9E	RES	3,(IY+d)	ED4D	RETI		DD9Ed	SBC	1,(IX+d)
CB9F	RES	3,A	ED45	RETN		FD9Ed	SBC	A,(IY+d)
CB98	RES	3,B	CB16	RL	(HL)	9F	SBC	A,A
CB99	RES	3,C	DDCBd16	RL	(IX+d)	98	SBC	A,B
CB9A	RES	3,D	FDCBd16	RL	(IY+d)	99	SBC	A,C
CB9B	RES	3,E	CB17	RL	A	9A	SBC	A,D
CB9C	RES	3,H	CB10	RL	B	9B	SBC	A,E
CB9D	RES	3,L	CB11	RL	C	9C	SBC	A,H
CBA6	RES	4,(HL)	CB12	RL	D	9D	SBC	A,L
DDCBdA6	RES	4,(IX+d)	CB13	RL	E	ED42	SBC	HL,BC
FDCBdA7	RES	4,(IY+d)	CB14	RL	H	ED52	SBC	HL,DE
CBA7	RES	4,A	CB15	RL	L	ED62	SBC	HL,HL
CBA0	RES	4,B	17	RLA		ED72	SBC	HL,SP
CBA1	RES	4,C	CB06	RLC	(HL)	37	SCF	
CBA2	RES	4,D	DDCBd06	RLC	(IX+d)	CBC6	SET	0,(HL)
DBA3	RES	4,E	FDCBd06	RLC	(IY+d)	DDCBdC6	SET	0,(IX+d)
CBA4	RES	4,H	CB07	RLC	A	FDCBdC6	SET	0,(IY+d)
CBA5	RES	4,L	CB00	RLC	B	CBC7	SET	0,A
CBAE	RES	5,(HL)	CB01	RLC	C	CBCO	SET	0,B
DDCBdAE	RES	5,(IX+d)	CB02	RLC	D	CBC1	SET	0,C
FDCBdAE	RES	5,(IY+d)	CB03	RLC	E	CBC2	SET	0,D
CBAF	RES	5,A	CB04	RLC	H	CBC3	SET	0,E
CBA8	RES	5,B	CB05	RLC	L	CBC4	SET	0,H
CBA9	RES	5,C	07	RLCA		CBC5	SET	0,L
CBAA	RES	5,D	ED6F	RLD		CBCE	SET	1,(HL)
CBAB	RES	5,E	CB1E	RR	(HL)	DDCBdCE	SET	1,(IX+d)
CBAC	RES	5,H	DDCBd1E	RR	(IX+d)	FDCBdCE	SET	1,(IY+d)
CBAD	RES	5,L	FDCBd1E	RR	(IY+d)	CBCF	SET	1,A
CBB6	RES	6,(HL)	CB1F	RR	A	CBC8	SET	1,B
DDCBdB6	RES	6,(IX+d)	CB18	RR	B	CBC9	SET	1,C
FDCBdB6	RES	6,(IY+d)	CB19	RR	C	CBCA	SET	1,D
CBB7	RES	6,A	CB1A	RR	D	CBCB	SET	1,E
CBB0	RES	6,B	CB1B	RR	E	CBCC	SET	1,H
CBB1	RES	6,C	CB1C	RR	H	CBCD	SET	1,L
CBB2	RES	6,D	CB1D	RR	L	CBD6	SET	2,(HL)
CBB3	RES	6,E	1F	RRA		DDCBdD6	SET	2,(IX+d)
CBB4	RES	6,H	CB0E	RRC	(HL)	FDCBdD6	SET	2,(IY+d)
CBB5	RES	6,L	DDCBd0E	RRC	(IX+d)	CBD7	SET	2,A
			FDCBd0E	RRC	(IY+d)	CBD0	SET	2,B
CBBE	RES	7,(HL)	CB0F	RRC	A	CBD1	SET	2,C
DDCBdBE	RES	7,(IX+d)	CB08	RRC	B	CBD2	SET	2,D
FDCBdBE	RES	7,(IY+d)	CB09	RRC	C	CBD3	SET	2,E
CBBF	RES	7,A	CB0A	RRC	D	CBD4	SET	2,H
CBB8	RES	7,B	CB0B	RRC	E	CBD5	SET	2,L
CBB9	RES	7,C	CB0C	RRC	H	CBD8	SET	3,B
CBBA	RES	7,D	CB0D	RRC	L	CBDE	SET	3,(HL)
CBBBB	RES	7,E	OF	RRCA		DDCBdDE	SET	3,(IX+d)
CBBC	RES	7,H	ED67	RRD		FDCBdDE	SET	3,(IY+d)

CODE OBJET	INSTRUCTION		CODE OBJET	INSTRUCTION		CODE OBJET	INSTRUCTION	
CBDF	SET	3,A	CBF3	SET	6,E	CB3E	SRL	(HL)
CBD9	SET	3,C	CBF4	SET	6,H	DDCBd3E	SRL	(IX+d)
CBDA	SET	3,D	CBF5	SET	6,L	FDCBd3E	SRL	(IY+d)
CBDB	SET	3,E	CBFE	SET	7,(HL)	CB3F	SRL	A
CBDC	SET	3,H	DDCBdFE	SET	7,(IX+d)	CB38	SRL	B
CBDD	SET	3,L	FDCBdFE	SET	7,(IY+d)	CB39	SRL	C
CBE6	SET	4,(HL)	CBFF	SET	7,A	CB3A	SRL	D
DDCBdE6	SET	4,(IX+d)	CBF8	SET	7,B	CB3B	SRL	E
FDCBdE6	SET	4,(IY+d)	CBF9	SET	7,C	CB3C	SRL	H
CBE7	SET	4,A	CBFA	SET	7,D	CB3D	SRL	L
CBE0	SET	4,B	CBFB	SET	7,E	96	SUB	(HL)
CBE1	SET	4,C	CBFC	SET	7,H	DD96d	SUB	(IX+d)
CBE2	SET	4,D	CBFD	SET	7,L	FD96d	SUB	(IY+d)
CBE3	SET	4,E	CB26	SLA	(HL)	97	SUB	A
CBE4	SET	4,H	DDCBd26	SLA	(IX+d)	90	SUB	B
CBE5	SET	4,L	FDCBd26	SLA	(IY+d)	91	SUB	C
CBEE	SET	5,(HL)	CB27	SLA	A	92	SUB	D
DDCBdEE	SET	5,(IX+d)	CB20	SLA	B	93	SUB	E
FDCBdEE	SET	5,(IY+d)	CB21	SLA	C	94	SUB	H
CBEF	SET	5,A	CB22	SLA	D	95	SUB	L
CBE8	SET	5,B	CB23	SLA	E	D6d	SUB	d
CBE9	SET	5,C	CB24	SLA	H	AE	XOR	(HL)
CBEA	SET	5,D	CB25	SLA	L	DDAEd	XOR	(IX+d)
CBEB	SET	5,E	CB2E	SRA	(HL)	FDAEd	XOR	(IY+d)
CBEC	SET	5,H	DDCBd2E	SRA	(IX+d)	AF	XOR	A
CBED	SET	5,L	FDCBd2E	SRA	(IY+d)	A8	XOR	B
CBF6	SET	6,(HL)	CB2F	SRA	A	A9	XOR	C
DDCBdF6	SET	6,(IX+d)	CB28	SRA	B	AA	XOR	D
FDCBdF6	SET	6,(IY+d)	CB29	SRA	C	AB	XOR	E
CBF7	SET	6,A	CB2A	SRA	D	AC	XOR	H
CBF0	SET	6,B	CB2B	SRA	E	AD	XOR	L
CBF1	SET	6,C	CB2C	SRA	H	EEd	XOR	d
CBF2	SET	6,D	CB2D	SRA	L			

ANNEXE 4

Code des instructions du Z 80

Dans cette partie les instructions sont classées suivant l'ordre croissant de leur code hexadécimal.

00	NOP	
01	LD	BC,+dddd
02	LD	(BC),A
03	INC BC	
04	INC B	
05	DEC B	
06	LD	B,+dd
07	RLCA	
08	EX	AF,A'F'
09	ADD	HL,BC
0A	LD	A,(BC)
0B	DEC	BC
0C	INC C	
0D	DEC C	
0E'	LD	C,+dd
0F	RRCA	
10	DJNZ	d
11	LD	DE,+dddd
12	LD	(DE),A
13	INC DE	
14	INC D	
15	DEC D	
16	LD	D,+dd
17	RLA	
18	JR	d
19	ADD	HL,DE
1A	LD	A,(DE)
1B	DEC DE	
1C	INC E	
1D	DEC E	
1E	LD	E,+dd
1F	RRA	
20	JR	NZ,d
21	LD	HL,+dddd
22	LD	(addr.) HL
23	INC HL	
24	INC H	
25	DEC H	
26	LD	H,+dd
27	DAA	
28	JR	Z,d
29	ADD	HL,HL
2A	LD	HL,(addr.)
2B	DEC HL	
2C	INC L	
2D	DEC L	

2E	LD	L,+dd
2F	CPL	
30	JR	NC,d
31	LD	SP,+dddd
32	LD	(addr.), A
33	INC SP	
34	INC	(HL)
35	DEC	(HL)
36	LD	(HL),+dd
37	SCF	
38	JR	C,d
39	ADD	HL,SP
3A	LD	A,(addr.)
3B	DEC SP	
3C	INC A	
3D	DEC A	
3E	LD	A,+dd
3F	CCF	
40	LD	B,B
41	LD	B,C
42	LD	B,D
43	LD	B,E
44	LD	B,H
45	LD	B,L
46	LD	B,(HL)
47	LD	B,A
48	LD	C,B
49	LD	C,C
4A	LD	C,D
4B	LD	C,E
4C	LD	C,H
4D	LD	C,L
4E	LD	C,(HL)
4F	LD	C,A
50	LD	D,B
51	LD	D,C
52	LD	D,D
53	LD	D,E
54	LD	D,H
55	LD	D,L
56	LD	D,(HL)
57	LD	D,A
58	LD	E,B
59	LD	E,C
5A	LD	E,D
5B	LD	E,E
5C	LD	E,H
5D	LD	E,L
5E	LD	E,(HL)
5F	LD	E,A
60	LD	H,B
61	LD	H,D
62	LD	H,C
63	LD	H,E
64	LD	H,H
65	LD	H,L
66	LD	H,(HL)
67	LD	H,A
68	LD	L,B
69	LD	L,C
6A	LD	L,D
6B	LD	L,E
6C	LD	L,H
6D	LD	L,L
6E	LD	L,(HL)
6F	LD	L,A
70	LD	(HL),B
71	LD	(HL),C
72	LD	(HL),D
73	LD	(HL),E
74	LD	(HL),H
75	LD	(HL),L
76	HALT	
77	LD	(HL),A
78	LD	A,B
79	LD	A,C
7A	LD	A,D
7B	LD	A,E
7C	LD	A,H
7D	LD	A,L
7E	LD	A,(HL)
7F	LD	A,A
80	ADD	A,B
81	ADD	A,C

82	ADD	A,D
83	ADD	A,E
84	ADD	A,H
85	ADD	A,L
86	ADD	A,(HL)
87	ADD	A,A
88	ADC	A,B
89	ADC	A,C
8A	ADC	A,D
8B	ADC	A,E
8C	ADC	A,H
8D	ADC	A,L
8E	ADC	A,(HL)
8F	ADC	A,A
90	SUB	B
91	SUB	C
92	SUB	D
93	SUB	E
94	SUB	H
95	SUB	L
96	SUB	(HL)
97	SUB	A
98	SBC	A,B
99	SBC	A,C
9A	SBC	A,D
9B	SBC	A,E
9C	SBC	A,H
9D	SBC	A,L
9E	SBC	A,(HL)
9F	SBC	A,A
A0	AND	B
A1	AND	C
A2	AND	D
A3	AND	E
A4	AND	H
A5	AND	L
A6	AND	(HL)
A7	AND	A
A8	XOR	B
A9	XOR	C
AA	XOR	D
AB	XOR	E
AC	XOR	H
AD	XOR	L
AE	XOR	(HL)
AF	XOR	A
B0	OR	B
B1	OR	C
B2	OR	D
B3	OR	E
B4	OR	H
B5	OR	L
B6	OR	(HL)
B7	OR	A
B8	CP	B
B9	CP	C
BA	CP	D
BB	CP	E
BC	CP	H
BD	CP	L
BE	CP	(HL)
BF	CP	A
C0	RET	NZ
C1	POP	BC
C2	JP	NZ,addr.
C3	JP	addr.
C4	CALL	NZ,addr.
C5	PUSH	BC
C6	ADD	A,+dd
C7	RST	00
C8	RET	Z
C9	RET	
CA	JP	A,addr.
CB		
CC	CALL	Z,addr.
CD	CALL	addr.
CE	ADC	A,+dd
CF	RST	08 H
D0	RET	NC
D1	POP	DE
D2	JP	NC,addr.
D3	OUT	(+dd)A
D4	CALL	NC, addr.
D5	PUSH	DE

D6	SUB	+dd
D7	RST	10 H
D8	RET C	
D9	EXX	
DA	JP	C,addr.
DB	IN	A,(+dd)
DC	CALL	C,addr.
DD	voir ci-après	
DE	SBC	A,+dd
DF	RST	18 H
E0	RET PO	
E1	POP HL	
E2	JP	PO,addr.
E3	EX	(HL),SP
E4	CALL	PO,addr.
E5	PUSH	HL
E6	AND	+dd
E7	RST	20 H
E8	RET PE	
E9	JP	(HL)
EA	JP	PE,addr.
EB	EX	DE,HL
EC	CALL	PE,addr.
ED	voir ci-après	
EE	XOR	+dd
EF	RST	28 H
F0	RET P	P
F1	POP AF	AF
F2	JP	P,addr.
F3	DI	
F4	CALL	PGddr.
F5	PUSH	AF
F6	OR	+dd
F7	RST	30 H
F8	RET M	
F9	LDS	SP,HL
FA	JP	M,addr.
FB	EI	
FC	CALL	M,addr.
FD	voir ci-après	
FE	CP	+dd
FF	RST	38 H

Instructions ED

ED 40	IN	B,(C)
ED 41	OUT	(C),B
ED 42	SBC	HL,BC
ED 43	LD	(addr.),BC
ED 44	NEG	
ED 45	RETN	
ED 46	IM 0	
ED 47	LD	I,A
ED 48	IN	C,(C)
ED 49	OUT	(C),C
ED 4A	ADC	HL,BC
ED 4B	LD	BC,(addr.)
ED 4D	RETI	
ED 4F	LD	R,A
ED 50	IN	D,(C)
ED 51	OUT	(C),D
ED 52		SBC HL,DE
ED 53	LD	(addr.),DE
ED 56	IM	1
ED 57	LD	A,1
ED 58	IN	E,(C)
ED 59	OUT	(C),E
ED 5A	ADC	HL,DE
ED 5B	LD	DE,(addr.)
ED 5F	LD	A,R
ED 60	IN	H,(C)
ED 61	OUT	(C),H
ED 62	SBC	HL,HL
ED 63	LD	(addr.),HL
ED 66	IM	2
ED 67	RRD	
ED 68	IN	L,(C)

ED 69	OUT	(C),L
ED 6A	ADC	HL,HL
ED 6B	LD	HL,(addr.)
ED 6F	RLD	
ED 72	SBC	HL,SP
ED 73	LD	(addr.),SP
ED 78	IN	A,(C)
ED 79	OUT	(C),A
ED 7A	ADC	HL,SP
ED 7B	LD	SP,(addr.)
ED A0	LDI	
ED A1	CPI	
ED A2	INI	
ED A3	OUTI	
ED A8	LDD	
ED A9	CPD	
ED AA	IND	
ED AB	OUTB	
ED B0	LDIR	
ED B1	CPIR	
ED B2	INIR	
ED B8	LDDR	
ED B9	CPDR	
ED BA	INDR	
ED BB	OTRD	

Instructions DD et FD

Nous ne donnons que les codes des instructions DD qui travaillent sur le registre d'index IX, les instructions travaillant sur le registre d'index IY sont les mêmes avec la différence suivante. Leurs codes commencent par FD au lieu de DD.

DD 09		ADD	IX,BC
DD 19		ADD	IX,DE
DD 21	+dddd	LD	IX,+dddd
DD 22	addr.	LD	(addr.),IX
DD 23		INC	IX
DD 29		ADD	IX,IX
DD 2A	addr.	LD	IX,(addr.)
DD 2B		DEC	IX
DD 34	d	INC	(IX+d)
DD 35	d	DEC	(IX+d)
DD 36	d+dd	LD	(IX+d),+dd
DD 39		ADD	IX,SP
DD 46	d	LD	B,(IX+d)
DD 4E	d	LD	C,(IX+d)
DD 56	d	LD	D,(IX+d)
DD 5E	d	LD	E,(IX+d)
DD 66	d	LD	H,(IX+d)
DD 6E	d	LD	L,(IX+d)
DD 70	d	LD	(IX+d),B
DD 71	d	LD	(IX+d),C
DD 72	d	LD	(IX+d),D
DD 73	d	LD	(IX+d),E
DD 74	d	LD	(IX+d),H
DD 75	d	LD	(IX+d),L
DD 77	d	LD	(IX+d),A
DD 7E	d	LD	A,(IX+d)
DD 86	d	ADD	A,(IX+d)
DD 8E	d	ADC	A,(IX+d)
DD 96	d	SUB	(IX+d)
DD 9E	d	SBC	A,(IX+d)
DD A6	d	AND	(IX+d)
DD AE	d	XOR	(IX+d)
DD B6	d	OR	(IX+d)
DD BE	d	CP	(IX+d)
DD CB	d 06	RLC	(IX+d)
DD CB	d OE	RRC	(IX+d)
DD CB	d 16	RL	(IX+d)
DD CB	d 1E	RR	(IX+d)

DD CB d	26	SLA	(IX+d)
DD CB d	2E	SRA	(IX+d)
DD CB d	3E	SRL	(IX+d)
DD CB d	46	BIT	0,(IX+d)
DD CB d	4E	BIT	1,(IX+d)
DD CB d	56	BIT	2,(IX+d)
DD CB d	5E	BIT	3,(IX+d)
DD CB d	66	BIT	4,(IX+d)
DD CB d	6E	BIT	5,(IX+d)
DD CB d	76	BIT	6,(IX+d)
DD CB d	7E	BIT	7,(IX+d)
DD CB d	86	RES	0,(IX+d)
DD CB d	8E	RES	1,(IX+d)
DD CB d	96	RES	2,(IX+d)
DD CB d	9E	RES	3,(IX+d)
DD CB d	A6	RES	4,(IX+d)
DD CB d	AE	RES	5,(IX+d)
DD CB d	B6	RES	6,(IX+d)
DD CB d	BE	RES	7,(IX+d)
DD CB d	C6	SET	0,(IX+d)
DD CB d	CE	SET	1,(IX+d)
DD CB d	D6	SET	2,(IX+d)
DD CB d	DE	SET	3,(IX+d)
DD CB d	E6	SET	4,(IX+d)
DD CB d	EE	SET	5,(IX+d)
DD CB d	F6	SET	6,(IX+d)
DD CB d	FE	SET	7,(IX+d)
DD E1		POP	IX
DD E3		EX	(SP),IX
DD E5		PUSH	IX
DD E9		JP	(IX)
DD F9		LD	SP,IX

ANNEXE 5

Code des caractères du SINCLAIR ZX 81

Code	*Caractère*	*Hex*	*Code*	*Caractère*	*Hex*
0	espace	00	24	/	18
1	▘	01	25	;	19
2	▝	02	26	,	1A
3	▀	03	27	.	1B
4	▖	04	28	0	1C
5	▌	05	29	1	1D
6	▞	06	30	2	1E
7	▛	07	31	3	1F
8	▒	08	32	4	20
9	▗	09	33	5	21
10	▜	0A	34	6	22
11	"	0B	35	7	23
12	£	0C	36	8	24
13	$	0D	37	9	25
14	:	0E	38	A	26
15	?	0F	39	B	27
16	(	10	40	C	28
17	)	11	41	D	29
18	⩾	12	42	E	2A
19	⩽	13	43	F	2B
20	=	14	44	G	2C
21	+	15	45	H	2D
22	–	16	46	I	2E
23	*	17	47	J	2F

Code	Caractère	Hex
48	K	30
49	L	31
50	M	32
51	N	33
52	O	34
53	P	35
54	Q	36
55	R	37
56	S	38
57	T	39
58	U	3A
59	V	3B
60	W	3C
61	X	3D
62	Y	3E
63	Z	3F
64	**RND**	40
65	**INKEY$**	41
66	**PI**	42
66 à		43 à
111	Inutilisés	6F
112	montée curseur	70
113	descente curseur	71
114	curs. vers la gauche	72
115	curs. vers la droite	73
116	**GRAPHICS**	74
117	**EDIT**	75
118	**NEWLINE**	76
119	**RUBOUT**	77
120	moke K L	78
121	**FUNCTION**	79
122	inutilisé	7A
123	inutilisé	7B
124	inutilisé	7C
125	inutilisé	7D
126	nombre	7E
127	curseur	7F
128	■	80
129	◪	81
130	◧	82
131	⬓	83
132	◨	84

Code	Caractère	Hex
133	◨	85
134	◪	86
135	◪	87
136	▩	88
137	▩	89
138	▩	8A
139	" inversé	8B
140	£ inversé	8C
141	$ inversé	8D
142	: inversé	8E
143	? inversé	8F
144	(inversé	90
145	) inversé	91
146	⩾ inversé	92
147	⩽ inversé	93
148	= inversé	94
149	+ inversé	95
150	— inversé	96
151	* inversé	97
152	/ inversé	98
153	; inversé	99
154	, inversé	9A
155	. inversé	9B
156	0 inversé	9C
157	1 inversé	9D
158	2 inversé	9E
159	3 inversé	9F
160	4 inversé	A0
161	5 inversé	A1
162	6 inversé	A2
163	7 inversé	A3
164	8 inversé	A4
165	9 inversé	A5
166	A inversé	A6
167	B inversé	A7
168	C inversé	A8
169	D inversé	A9
170	E inversé	AA
171	F inversé	AB
172	G inversé	AC
173	H inversé	AD
174	I inversé	AE

Code	*Caractère*	*Hex*
175	J inversé	AF
176	K inversé	B0
177	L inversé	B1
178	M inversé	B2
179	N inversé	B3
180	O inversé	B4
181	P inversé	B5
182	Q inversé	B6
183	R inversé	B7
184	S inversé	B8
185	T inversé	B9
186	U inversé	BA
187	V inversé	BB
188	W inversé	BC
189	X inversé	BD
190	Y inversé	BE
191	Z inversé	BF
192	""	C0
193	**AT**	C1
194	**TAB**	C2
195	inutilisé	C3
196	**CODE**	C4
197	**VAL**	C5
198	**LEN**	C6
199	**SIN**	C7
200	**COS**	C8
201	**TAN**	C9
202	**ASN**	CA
203	**ACS**	CB
204	**ATN**	CC
205	**LN**	CD
206	**EXP**	CE
207	**INT**	CF
208	**SQR**	D0
209	**SGN**	D1
210	**ABS**	D2
211	**PEEK**	D3
212	**USR**	D4
213	**STR$**	D5
214	**CHR$**	D6
215	**NOT**	D7

Code	*Caractère*	*Hex*
216	**	D8
217	**OR**	D9
218	**AND**	DA
219	⩽=	DB
220	⩾=	DC
221	⩽⩾	DD
222	**THEN**	DE
223	**TO**	DF
224	**STEP**	E0
225	**LPRINT**	E1
226	**LLIST**	E2
227	**STOP**	E3
228	**SLOW**	E4
229	**FAST**	E5
230	**NEW**	E6
231	**SCROLL**	E7
232	**CONT**	E8
233	**DIM**	E9
234	**REM**	EA
235	**FOR**	EB
236	**GOTO**	EC
237	**GOSUB**	ED
238	**INPUT**	EE
239	**LOAD**	EF
240	**LIST**	F0
241	**LET**	F1
242	**PAUSE**	F2
243	**NEXT**	F3
244	**POKE**	F4
245	**PRINT**	F5
246	**PLOT**	F6
247	**RUN**	F7
248	**SAVE**	F8
249	**RAND**	F9
250	**IF**	FA
251	**CLS**	FB
252	**UNPLOT**	FC
253	**CLEAR**	FD
254	**RETURN**	FE
255	**COPY**	FF

ANNEXE 6

Adresses utiles

L'assembleur présenté au chapitre 7 est un produit de:

— BUG-BYTE
100 THE ALBANY
OLD HALL STREET
LIVERPOOL L3 9EP
GRANDE-BRETAGNE

Ce programme est distribué en France par:

— DIRECO INTERNTIONAL
30 Avenue de Messine
75008 PARIS

— GOAL COMPUTER
15 Rue St-Quentin
75010 PARIS

Imprimerie de la Manutention à Mayenne
Dépôt légal : Janvier 1983
N° d'éditeur : 3892

Avec "Langage machine, trucs et astuces sur ZX 81", obtenez encore plus de votre micro-ordinateur SINCLAIR ZX 81. Cette lecture vous apprendra comment scruter le clavier, comment réaliser vos propres jeux avec des graphiques animés extrêmement rapides !... Vous y trouverez aussi comment générer une instruction REM de plusieurs K octets en quelques secondes !... De plus un programme d'aide à la mise au point des programmes écrits en langage machine y est clairement détaillé. "Langage machine, trucs et astuces sur ZX 81" vous apportera une aide vraiment efficace dans la mise en œuvre du langage machine sur votre ZX 81.

EYROLLES

EYROLLES

LANGAGE MACHINE, TRUCS ET ASTUCES SUR ZX 81

G. NOLLET

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/